Smart Engineering System Design, vol. 4:243-252, 2002

NETWORK CONGESTION ARBITRATION AND SOURCE PROBLEM PREDICTION USING NEURAL NETWORKS

J. ALAN BIVENS
Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
bivenj@cs.rpi.edu

BOLESLAW K. SZYMANSKI
Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
szymansk@cs.rpi.edu

MARK J. EMBRECHTS
Decision Sciences And Engineering Systems
Rensselaer Polytechnic Institute
Troy, New York 12180-3590
EMBREM@RPI.EDU

ABSTRACT
Rapidly growing needs for networking in the Internet and in intranets make network management increasingly important in today's computer world. We propose using learning techniques to predict network congestion problems before they start impacting the performance of services. In this paper we focus on using a simple feedforward neural network to predict severe congestion in a network. We also use neural networks to predict the source or sources responsible for the congestion, and we design and apply a simple control method for limiting the rate of the offending sources so that congestion can be avoided. Unlike the usual TCP/IP flow control, the proposed method is applied only to selected nodes and converges to the final rate faster. The described techniques set the stage for a new wave of network managers that are capable of preventing networking problems instead of repairing them.

KEYWORDS: congestion control, neural networks, weight pruning, network problem prediction, flow rate control

INTRODUCTION

In the growing world of networking, more and more emphasis is being placed on speed, connectivity, and reliability. Network performance is vital to businesses for inter-business operations as well as bringing a product to the consumers through electronic commerce. Networking has influenced our everyday lives, as the interconnectivity of families and friends has changed the ways in which they communicate and seek information.

With the growing number of network users, more emphasis is placed on the maintenance and reliability of the networks. When network problems occur, they often catastrophically break the service for those enterprises or individuals that depend on the network connection. Sometimes, such breaks of service are just annoying, but for companies and commercial users they often mean lost revenues on the order of thousands, or even millions, of dollars. Such breaks have become a significant problem in all forms of electronic commerce.

To address this difficulty, a system is needed to insure network availability and efficiency by preventing such costly network breakdowns. The first step towards this end is to create a system with the intelligence to recognize, as early as possible, early signs of incoming network service difficulties. If the problem can be recognized in advance, changing network parameters can possibly circumvent the problem.

Today, many companies provide a sophisticated suite of network performance measurement tools and alarm-raising systems to try to address the problem. However, these tools simply give the network administrator a great deal of information about the problem after it has happened, hoping this will help the administrator find the cause of the problem and suitable solution quickly. None focus on learning from the past breakdowns how to detect the problem beforehand. Early detection is of course preferable, because it would allow the administrator or the program itself to prevent the problem from ever occurring.

We propose a system that uses neural networks to detect network congestion before it results in a breakdown of the network service and which also identifies the source of the congestion. Having the nodes identified, our system applies the flow rate restriction adaptively to the identified sources to avoid congestion overflowing the router's buffers. It should be noted that design and experiments presented in this paper focus on congestion control; however, the techniques could be applicable to other network problems. It should also be noted that the remedy in the form of flow rate restrictions can be applied directly to the original flow source if it is within the domain controlled by our system, or it could be applied to the edge router to the domain to which our system is applied. In the latter case, the restriction will result in the packets of the restricted flow being dropped at the edge router to the domain. This kind of a solution in which the congestion is decomposed and "moved" from the internal routers to the edge routers is becoming increasingly popular in modern traffic management. Finally, it should be noted that in such edge-control techniques the domains controlled by separate systems will collaborate through the edge routers. Dropping packets at the entry edge router of one domain will cause the packets to be dropped at the exit router of the neighboring domain which will treat such dropping as congestion and then will identify the source of the flow. As a result, our techniques can be applied locally at a domain of the decomposed network and their congestion solution will iteratively be mapped to the corresponding edge routers of the intermediate domains until the source of the flow is found and informed of the need to decrease the traffic.

Another remark is needed to relate the present work to the differentiated services, methods of creating different levels of services for customers willing to pay higher levels. As more products are created to control networks, differentiated services will become very important for the Internet. Identification of sources that can be forced to limit their flow rate can lead to accounting for different priorities of traffic and offending flows. For example, if traffic from a certain machine is deemed high priority, the system may restrict other machines, instead of slowing down the high priority machine. Changing the architecture to operate on a flow basis instead of a machine basis could also be easily done to account for a variety of traffic from each machine, each with a different priority. Hence, the techniques that we present in this paper are directly applicable to the "best effort traffic" over the Internet, but their extensions to Differentiated Services or Quality of Services environments are straightforward.

RELATED WORK

Actively managing networks remains an open topic of research with many interesting and difficult challenges. One of the challenges deals with how relevant traffic information is handled through the network. There is a great deal of research being done on ways to have each node receive all relevant information as packets go through the network. Most research and development in this area focuses on active networks. Active networks allow each packet of information to contain some computation that is executed on each node that the packet reaches in its travel trough the network. Hence, in active networks, each packet contains both data and code. The code contained in the packet is executed in an execution environment determined by the node's operating system. This code could consist of function calls, or actual code to be compiled in the executing environment. Using active networks, the network can perform customized computations on the data flowing through the network [15].

Some believe the real power of active networks is not in their computational ability, but in communication ability. These networks may focus on Quality of Service issues and network traffic classification. Active networks with this focus can monitor and regulate themselves with every packet flowing through the network. They can also insure that certain traffic will always receive reliable service [6]. Many times these active networks can even be used to augment the protocols used to send the data, providing additional efficiency in the data sending process [1]. Using active networks could certainly afford us the ability to easily optimize the efficiency of the network. However, the cost of an implementation such as active networks is very high. They require resources on every node in the network for every packet that flows through the network. Each packet must take time for an execution environment to be designated and its code to be executed at every node. There are also security problems in this approach, because every node in the network may execute the code of every packet. In this case, special security measures would have to be taken at every node to avoid executing code from a malicious packet.

Some implementations of active networks involve limiting the "active nodes" to those surrounding special links. Typically, these are links selected because of historically poor bandwidth capabilities or tendencies to drop packets. The purpose, in these cases, is to change the traffic (split packets, or send part of flow through different routes.) These implementations are called Transformer Tunnels [3]. Whereas these tunnels do limit the active network computational liabilities to a smaller number of nodes, they do not give us the power to look at statistics from other locations in the network. A neural network agent, acting as a controller, can gather the necessary statistics and make predictions, while avoiding many of the problems of active networks.

Approaching networking issues with neural networks is not uncommon. A group at Prairie View A&M University developed a neural network-based system, which looks at a network's traffic pattern and proposes a new physical configuration of the network. Their goal was to reduce the amount of data flowing around the network, thereby decreasing the response time, and reducing the error rate [11].

Other neural network research has been done in ATM networks with Connection Admission Control (CAC) schemes. In these works, the authors use fast converging neural networks to predict cell loss in a switch using data from a cell bouncing between two switches [16]. In ATM networks, a switch is a machine which receives data from one interface and quickly forwards it to another interface closer to the destination desired. The cell is simply the name given to the piece of data because it is of a uniform size and structure. This effort involved a degree of loss prediction, but offers no solution for determining the problem source or solution.

Another group at the University of Maryland is using neural networks to route traffic in Multistage Interconnection Networks [10]. Routing can be static or dynamic. Static routing involves an administrator manually configuring each router to forward packets of a certain destination along a certain interface, while dynamic routing usually uses a distributed algorithm and special protocols to determine the best routes. Authors at Maryland use a neural network that functions as a very robust parallel computer, dynamically generating routes faster than conventional routing approaches. The robust nature of neural networks was one of the factors that attracted the authors to this approach.

A group at Rensselaer Polytechnic Institute is doing very similar research [17] [18]. They detect changes in traffic patterns through the use of a sequential Generalized Likelihood Ratio (GLR) test. They first gather time series MIB variable data using SNMP. This data is then split into windows of 2.5 minutes each to create piecewise stationary auto-regressive models. Using these windows, a (GLR) sequential hypothesis test is performed to determine the extent of statistical deviation between two adjacent time windows. Once changes are detected using the GLR, the authors explored two ways of correlating the different alarms between values of the several MIB variables. They first try a Bayesian belief network whose model is based on a directed graph that spatially represents the hierarchical structure of the MIB variables. Their second technique was a duration filter which would correlate the propagation of many alarms with the MIB's internal variable dependencies during a certain duration period. Their algorithm was "trained" or optimized using five of nine fault data sets, and proved general enough to detect three out of the remaining four fault data sets. However, this group used a hypothesis testing method for detecting patterns, while we propose a neural network to learn the patterns leading to network faults. We also focus on predicting problem sources and correcting the problem in a timely fashion, whereas their work was just in general detection.

Another group with similar research goals can be found at the University of Michigan [7]. There is work being done there in congestion control and mitigation strategies. However, this group uses a queuing theory approach rather than a neural network to guide the mitigation efforts. And their mitigation efforts consist of rerouting the traffic whereas ours focuses on implementing a control architecture.

Wu Chang Feng, also of the University of Michigan, has developed a neural network optimizer for bandwidth allocation in telecommunications networks [9]. In Feng's work, feed-forward neural networks were used to quickly optimize bandwidth allocation. To evaluate the performance of the neural network optimizer, the results were compared to results from a linear programming optimizer. Feng removes the real-time control effect from this work, allowing the neural network to function more as a traditional optimizer. Rather than assigning an appropriate bandwidth to the nodes across the network using a neural network optimizer, we use a neural network predictor as a pure control agent, regulating real-time traffic on the network. This allows the network to function as normal with our neural network watching real-time statistics to determine if they indicate the start of a networking problem.

Neural networks have also been used to decide if certain requests will satisfy the Quality of Service (QoS) parameters set by the network administrators [12]. QoS usually involves classifying network traffic and setting certain restrictions on the traffic based on its class. The predictive powers of neural networks are used to predict if QoS standards will be upheld with the entry of new traffic. The authors of [12] do not address predicting congestion causes, nor do they need to make any attempt towards correction. Their solution falls in the category of open-loop solutions, those that prevent problems by not allowing the network to enter into any state in which a transition into a problem state is possible. Limiting network activities to states which could only transition to positive states can vastly reduce the utilization of the network, given the unpredictability of networking traffic.

ARCHITECTURE

A high level view of our architecture reveals a network with a control agent existing somewhere on a node in that network. This control agent has both the power to read from and to influence network nodes. The nodes involved would either report the necessary statistics to the control agent or the control agent would poll these nodes.

Data Network

Optimally, we would have tested the system on a local computer lab or a testing lab put together for this purpose. However, neither of these was available at the time of project development. In the absence of the needed testbed, NS, a discrete-event network simulator targeted at networking research, was used to model the network and different scenarios of network traffic (NS can by found and downloaded from http://www.isi.edu/nsnam/ns/) []. NS simulates network architectures on a packet by packet basis, giving the user the ability to monitor very specific as well as aggregate statistics about all facets of the network. This, of course, made the integration of a control agent easier, but a similar design could be implemented on a real network.

In our example, the network consisted of several nodes in a configuration where all of the network nodes were attempting to send data to one node (see Figure 1(a)). Each node attempts to send at a random bit rate. A random amount of variance is given to each node's rate to better represent traffic in a real network and possible traffic coming in from other nodes outside of our simulation. Link capacities between sending nodes were given arbitrary values (described in a later section) for testing purposes. Some links were able to handle much more traffic than other links.

Control Agent

We create a control agent containing a neural network that is trained prior to being placed in production. In our simulation, the control agent is called at a regular interval in part of the simulation code. This enables the agent to easily monitor and influence traffic statistics from each node. The control agent gathers information from each managed node, performs several mathematical functions normalizing the values, and makes a decision about where, if anywhere, network problems will occur. With the predicted problem in our grasp, we can take steps to stop or prevent it.

Implementation Details

The system consists of three separate programs, one implemented in Tcl, the language used to run simulations in NS, one implemented in C for file manipulation, and a third actually running the neural network. We use the publicly available MetaNeural Neural Network application as our neural network (MetaNeural can be obtained from http://www.drugmining.com/). MetaNeural is a general-purpose backpropagation program. These programs communicate with each other via files to synchronize the running of NS with the running of the neural network program, determining if a current network configuration might cause a problem. An illustration of these relationships is shown in Figure 2(a and b).

The Simulation Network

The simulated network is arranged such that six sending nodes are connected to one receiving node through several links which direct the packets to the destination (Figure 1(a)). The sending nodes produce data in a way similar to Universal Datagram Protocol (UDP) agents, sending constant bit rate (CBR) traffic with a randomized parameter to add variance to the traffic. In NS, each connection is explicitly stated and each sending agent in each node is configured to send to a particular receiving agent. To determine how fast the sender sends data, the packet size and a packet interval are given in the simulation script that defines the simulation run. The sender sends a packet of the designated size at the designated interval. The receiver simply has a null agent that receives the data and sends no responses. NS Queue Monitors are attached to the queues to keep track of the status of each queue. We gather statistics such as packets received and the size of the queue during the simulation. During the simulation, the control agent executes at a polling interval, monitoring the traffic and making decisions. Files are created for each node to keep track of that node's data. During each run of the network simulator, the files are extended with the new data from the latest interval. The most important part of the control agent is the neural network prediction module. For our agent implementation, we used a single hidden layer, feed-forward neural network. This was a compiled application, so wrappers were needed to control the input and output dealing with the neural network. The wrapper program was written in C and is called after the data files are updated by the simulator. The simulator halts until the C program finishes.

C Wrapper: As mentioned before, to execute the control agent, a C wrapper is first called. This is where the bulk of the calculations for the neural network program are done. It first opens the files written by the simulator which contain historical and current values for the number of packets. The program uses these values and computes the average number of packets, the variance of packets, and the third momentum given the appropriate polling interval. In the first iteration, the average is the current number of packets and the variance and third moment are zero. These values are then normalized for the neural network using a basic normalization function. The normalized values are then combined into one input file to the neural network package for a decision. The neural network program is executed using new input files and the output is rendered in yet another file. This file is read by the C wrapper and converted into a readable format for the simulator to process.

Neural Network Specifics: The neural network used by the control agent has 3*n input nodes, 1 hidden layer containing n nodes, and n nodes in the output layer. The 3*n input nodes correspond to the n traffic generating nodes in the network simulation; there are three input nodes for each node in the network simulation corresponding to the average number of packets, variance, and third moment for each monitored node. In our example (Figure 1(a)), n would be six because only six nodes were actually contributing traffic to our network.

To further stress the importance of adjacency relationships between nodes in the data network, we placed an additional optimization of the structure of the neural network. The weights were pruned to the point in which the neural network reflected the connectivity of the actual network. The n nodes in the hidden layer also represent active nodes in the data network. Instead of providing a fully connected environment between the input layer and the first hidden layer, we only allowed connectivity from input neurons that represent nodes adjacent to represented nodes in the hidden layer. An example is shown in Figures 1(a and b).

In the model of the neural network in Figure 1(b), the hidden layer is representative of the participating nodes in the data network. The statistical data regarding each node is provided to the hidden node representing the actual node as well as to the hidden nodes representing the actual node's neighbors. This is continued for all first layer nodes of the neural network. As a result, the statistical information from node 1 is given to both the hidden node representing node 1, and the hidden node representing node 5 (1's neighbor.) This process is important in realizing the relationships between adjacent nodes in a data communications network.

The output of the neural network is a mask representation of which nodes have been suspected of causing the problem. For example, an output of "010000" would mean the second node in the network was responsible for the network congestion. An output of "010100" would mean the second and forth nodes were both to blame, and "000000" would mean no problem threat has been detected.

The neural network was trained off-line, which involved creating a pattern file from which the neural network would learn about congestion. Eighty-eight patterns were used to train the network; half were samples containing no network problems and half had congestion problems at various locations. In training the neural network, early stopping was used, allowing the training to go for about 15000 iterations. In this case, the least squares error was equal to or less than 0.04%.

Control from the Agent: As stated before, in NS, both an interval and a packet size are provided for agents sitting at the sending nodes to determine bandwidth used. The agent will send one packet at every interval, therefore the smaller the interval, the higher the bit rate. If our neural network predicts that a particular node will be responsible for congestion, we conclude that the predicted problem source is using too many resources. To correct the predicted cause node traffic rate, we add to its sending interval a small Δt, thus reducing its bit rate. For example if node 1 was predicted as the problem source, Equation 1 would explain how node 1's bit rate would be corrected:

$$\text{Interval}_1 = \text{Interval}_1 + \Delta t \qquad (1)$$

This delta was chosen to be small with respect to the simulation time scale, because we do not want to take the chance of over-correcting or even worse, applying a large correction to the wrong node if our prediction was wrong. To take into account the small Δt, the interval in which our control agent executes also is relatively small. Therefore many of these small corrections can be applied which corrects a problem slowly without drastically changing any one node's level of service.

TEST CASES

Testing Environment

The tests were performed on a Sun Ultra-SPARC 10 running Solaris 5.6. The packet size was set at 500 bytes for each sending node; most of the links between nodes were set to 5 MB/s; and the link between nodes 5 and 6 was set at 10 MB/s. The delay for the links was set at 10ms and all queues implemented Stochastic Fair Queuing (Equally Fair Queuing). The interval was varied between 0.00100 and 0.00200 seconds (if the interval is 0.00100, a packet is sent every

0.001 seconds). The traffic generated is constant bit rate traffic, but the random parameter was set for each sending node so that the traffic would not be totally constant.

RESULTS

A general breakdown of the results can be found in the graph of Figure 3 which shows our current application detects and corrects congestion in about 90% of the cases. Our tests include cases in which corrections to one node are required, corrections to multiple nodes are required, and some where no correction is required. Failing includes either missing congestion or predicting congestion when there is none.

We ran thirty-one network simulations. Roughly 33% of the cases were simulations of a network without congestion problems. In these cases we would want our control agent to realize that it does not have to do anything. The detector realized that there was no correction needed in all but 1 case. In this isolated case our agent unnecessarily applied a single small correction to a single node. The correction that the control agent applied was with a single Δt, and therefore was minimal.

About 66% of the total cases had various levels of congestion in various locations in the network. Of the congested cases, we were able to predict the cause and fix the problems 85% of the time. The times we were able to detect the congestion problems can be found in Figure 4. Figure 4 shows that 60% of the time we detected congestion, we were able to fix the problem before packets were dropped in the network. These were truly remarkable results, because the congestion was completely eliminated before it occurred. In the cases that we could not stop packets from dropping, we were able to return the network to a stable state within 3.5 seconds after packets began to drop.

Finally, in the cases that our detector missed the threat of congestion there was a common characteristic. The neural network had trouble detecting congestion when a single node in a particular part of the data network caused a problem. This probably can be improved upon close examination of the training patterns and structure of the neural network.

COMPARISON TO MODERN TECHNIQUES FOR CONGESTION CONTROL

To evaluate the advantages of a system like the one we have described, we analyze below the current methods of congestion control. Today, most of the congestion control is handled in the transport layer of network communications. The most often used protocol for network transport in today’s applications is TCP (Transmission Control Protocol). TCP uses a combination of several algorithms to control congestion but only those that are relevant to our technique will be discussed here.

TCP Description

TCP congestion control is flow-based, meaning that the only information considered by the protocol concerns the sender and the receiver of a particular flow. TCP keeps track of several state variables for each connection or flow that it controls, and adjusts the state variables through two algorithms, slow start and congestion avoidance. Slow start is used at the beginning of a TCP flow, until congestion is detected. After congestion is found, TCP then switches to congestion avoidance.

The two most important variables that TCP maintains are the congestion window, *cwnd*, and the slow start threshold, *ssthresh*. The *cwnd* controls and limits the sender as to how much data the source can send at a given time before an acknowledgement is received. The *ssthresh* controls how long TCP will stay in slow start mode (predicted start of congestion). Selection of an appropriate value for the *cwnd* is significant because if it is too high, data can be lost quickly but a value that is too small leads to underutilization of the network. Because of the importance of this variable, the TCP source must set the *cwnd* based on the level of congestion that it perceives to exist in the network. However, the level of congestion in the network can increase and decrease with changes in the network. The short descriptions below show how slow start and congestion avoidance, consistantly alter the two variables as congestion changes in the network.

Slow Start: At the beginning of a connection, the *cwnd* is set to one segment and the *ssthresh* is set to a large number. The source sends the number of segments dictated by *cwnd*, and then waits for the segments to be acknowledged by the receiver. If all segments are acknowledged, the value of *cwnd* effectively doubles. The congestion window doubles because the source thinks there is no congestion in the network and determines it can send more data before waiting for an acknowledgement to further utilize the network. As long as the segments are all acknowledged, the *cwnd* will continue to double until *cwnd* >= *ssthresh*. When this value is reached, the source thinks that congestion can happen soon, so it switches to the congestion avoidance algorithm. If a segment is not acknowledged before some timeout occurs, the source views this as a significant sign of congestion. When the acknowledgement timeout has happened, *ssthresh* is set to half the value of *cwnd*, and *cwnd* is reset to one. However, the source remains in the slow start algorithm until the *cwnd* >= *ssthresh.*

Congestion Avoidance: Once *cwnd* >= *ssthresh*, TCP switches to the congestion avoidance algorithm to advance the *cwnd* slowly to utilize the network, but not cause congestion. To this end, congestion avoidance uses a process called additive increase/multiplicative decrease. If all segments of a *cwnd* are acknowledged by the receiver, the *cwnd* now increases by 1 instead of doubling. This additive increasing of *cwnd* continues until an acknowledgement is not received. When an acknowledgement timeout occurs, the source believes once again that it has caused congestion, sets *ssthresh* to half the value of *cwnd* (multiplicative decrease), and resets *cwnd* to 1 segment. Because *cwnd* would now be less than *ssthresh* again, TCP reverts back to the slow start algorithm until *cwnd* >= *ssthresh* [8].

An interesting point is that in additive increase/multiplicative decrease the source will reduce its congestion window much faster than it will increase it. If the congestion window increases, as quickly as it decreases (multiplicative increase/multiplicative decrease) then congestion would occur much faster and there would be many more dropped packets. It has been shown that the additive increase strategy is a "necessary condition for a congestion control mechanism to be stable." [2]

How our application is different: One of the biggest criticisms of TCP's method of control is that it can determine congestion only when data already have been lost. Our strategy determines that congestion is brewing, often before the congestion has actually happened.

Another big difference is revealed in the correction procedures. When TCP detects congestion by dropped packets, it reduces the sender's congestion window by half. Situations could arise where the node that is actually causing the congestion does not have packet loss, but another node along the way that sends out relatively low traffic could be the one to have an acknowledgement timeout. In the case just mentioned, the node for which the congestion window already was small becomes even smaller, in effect, punishing the wrong node. With the growing popularity of open source operating systems, users also easily could create "unruly"

TCP clients that do not adhere to the protocol rules creating havoc for all other “well-behaved” flows in a similar fashion. However, with our system, correction will be applied to a node or group of nodes causing the congestion. That node will then have a lower threshold placed on its traffic rate, forcing it to slow down.

FUTURE WORK

Extending Architecture

An extended architecture can be formed to suit the biggest communication networks by decomposing these large networks into domains small enough to influence easily. Most large networks are already composed hierarchically of domains that are collections of local networks. When the larger networks have been divided, a separate protocol or addition to current protocols can be used to determine domain-to-domain agreements.

As illustrated in Figure 5, the learning Control Agent would be somewhere within each domain defined in the network. Using the forecasting and detective powers of the agent, each domain would be regulated and operating at safe levels.

To negotiate any problems between domains, the control agents communicate with each other as to the effects of one domain on another. The control agents take this information into account just as they take the information from local nodes into account. The inter-domain negotiation will promote the same safe state of operations between domains, as it will inside each domain. In particular, a border router between two domains is a source in one and the destination in the other. If the flow creates the congestion in the second domain, the agent in control of the second domain will apply corrections to the source border router, causing the router to drop packets from the flow to enforce the lower rate allocated to the flow. The dropping of packets at the border router, which is a destination for the first domain, will be perceived as congestion by the agent controlling the first domain. As a result, the agent of the first domain will apply correction to the source of the traffic, thereby eliminating congestion. An important feature of this architecture is that if the agent has no control over the source, it will apply rate correction to the first router under its control causing it to drop packets from the offending flow. As a result, our technique will remedy malicious attacks by the “unruly” tcp sources by dropping the packets not responding to the request for the lower rate of traffic from the source.

Broader Detection

At present, our system only predicts and detects traffic congestion. It would be useful to create a system that can do the same for other problems. It would also be of great value to train the network over many different types of traffic.

Due to the changing nature of networking, it is not unreasonable for the network, at some point, not to fit the same mold our neural network learned. For this reason, it could prove beneficial to train the network with a real time recurrent learning algorithm to allow the neural network to continue learning after the training period [19]. Alternatively, we can envision an architecture in which a current generation of the neural network is used for control while simultaneously a new generation is being trained on the current traffic patterns. We believe that the period of run/training could be quite infrequent, something like few weeks at a time. Different control neural networks could be trained for different times, special events, and other cases where specialized traffic needs are noticed.

Application of Positive Corrections to Optimize Network

The present system predicts network congestion and takes steps to prevent it by forcing nodes to slow the sending rates. It would be equally beneficial to train the network also to detect conditions in which the current bandwidth thresholds are too strict for proper network utilization. In these cases, the system should relax the bandwidth threshold for selected nodes. We formed a direct comparison between TCP and our system using the File Transport Protocol (FTP) to transfer files from one node to the other. When transferring small files, our system always beats TCP. When transferring large files, TCP has a chance to take advantage of its additive increase and slow start techniques [2], and transfers the information faster than our strategy. However, TCP has the advantage of adding and subtracting until it can find an optimal bandwidth, while our present system only subtracts from the current rate. Therefore, adding a capability of applying a positive correction to selected nodes would make our system competitive with TCP also for transfer of large files.

Implementation of the System on a Real Network

Lastly, it would be most interesting to implement this system on a real network to see if similar results can be obtained. To apply to a real system, we could use Simple Network Management Protocol (SNMP) traffic variables []. SNMP monitors many statistics including several on the internet protocol and interface levels for a particular machine. Our traffic data would be functions of current data counts from the routers in the network.

Several avenues are available to control sources in the network. Classifications and service level agreements (SLAs) or communication pipes could be used to implement the control aspect. If either of SLAs or communication pipes are used, the control agent must reside on a node in the network that has access to all nodes being monitored. The close proximity keeps our control agent local and would prohibit implementation in a very large network without changing the architecture. A more interesting choice would be to implement our system as an Active Queue Management scheme.

In active queuing, some process is used to detect congestion "early." When congestion is detected, the sending protocol is notified to slow down through TCP's multiplicative decrease feature. Either a packet is dropped early to make the sender slow down or a process called Early Congestion Notification is used to set a special bit in the acknowledgement packet letting the sender know that the early congestion detection mechanism has detected early signs of congestion. When the sender gets an acknowledgement packet with this bit set, it will reduce its congestion window as if a packet has been dropped. The advantage of this technique is that it uses existing protocol techniques to control the flow without losing any data packets.

CONCLUSIONS

In this work, we set out to show that a neural network is a viable method of implementing a learning mechanism for data communication networks. We have illustrated, through the use of a network simulator, that a neural network can achieve great accuracy in predicting one particular network problem, namely congestion. We realize many more problems exist that for which this approach is applicable, but predicting congestion is just the first step towards our research goals. We also have shown one situation in which a carefully constructed neural network can achieve above average results when structural information about the actual data network is used to form the connections between layers of the neural network. This special design forces the neural network to consider the relationships only of those nodes that we think are important.

A learning mechanism can be of great value for a network manager. The generalization power of a neural network particularly is appropriate because of the unpredicted variance of parameters that the network manager encounters. Neural networks are an appropriate mechanism for decision making in pro-active network management and should be the subject of more research.

ACKNOWLEDGEMENT
The first author was supported by an IBM Graduate Fellowship during the writing of this paper.

The authors wish to express their gratitude to Michelle Conway for her help in implementing and running some of the experiments.

REFERENCES

[1] Alexander, D. Scott. (1998) The switchware active network architecture. IEEE Network Special Issue on Active and Controllable Networks, 12(3):29-36.

[2] Allman, M., Paxson, V., and Stevens, W. (1999) TCP Congestion Control. Request for Comments (RFC): 2581. April 1999.

[3] Badrinath, B.R. and Sudame, Pradeep. (1998) Transformer tunnels: A framework for providing route-specific adaptations. In The USENIX Annual Technical Conference. USENIX.

[4] Bagel, Prasad et al. (2000) “TCP Rate Control,” Computer Communication Review, Jan, vol. 30, num=1.

[5] Bajaj, Sandeep et al. (1999) “Improving Simulation for Network Research,” Technical Report 99-702, University of Southern California.

[6] K. L. Calvert. (1999) Architectural framework for active networks. RFC Draft.

[7] Campbell, Peter K., Christiansen, Alan, et al. (1997) Experiments with simple neural networks for real-time control. IEEE Journal on Selected Areas in Communications, 15(2):165-178.

[8] Clark, Dave. (1996) Computer Networking: A Systems Approach. Morgan Kaufmann Publishers.

[9] Feng, Wu Chang. (1999) Improving Internet Congestion Control and Queue Management Algorithms. PhD thesis, The University of Michigan, Ann Harbor, Michigan. Computer Science and Engineering.

[10] Giles, Lee and Goudreau, Mark. (1994) Routing in optical multistage interconnection networks: a neural network. Technical Report CS-TR-3227, University of Maryland.

[11] Murtaza, Mirza and Shah, Jayneen. (1995) Reducing data movement in client/server systems using neural networks. http://www.informs.org/Conf/NO95/TALKS/MA19.3.html.

[12] Ogier, Richard, Plotkin, Nina T., and Khan, Irfan. (1996) Neural network methods with traffic descriptor compression for call admission control. In Proceedings of the Conference on Computer Communications. IEEE Infocom.

[13] Ramakrishnan, K and Floyd, S. (1999) “A Proposal to add Explicit Congestion Notification (ECN) to IP,” Request for Comments: 2481.

[14] Stallings, William. (1999) “SNMP, SNMPv2, SNMPv3, and RMON1 and 2,” 3 ed. Addison Wesley Longman, Inc., Reading, Massachusetts.

[15] Tennenhouse, David L. and Wetherall, David J. (1996) Towards an active network architecture. Technical Newsletter: Computer Communication Review.

[16] Tham, C.K. and Soh, W. S. (1998) Multi-service connection admission control using modular neural networks. In Proceedings of the Conference on Computer Communications, page 1022, San Francisco, California, IEEE Infocom.

[17] Thottan, Marina and Ji, Chuanyi. (1998) Adaptive thresholding for proactive network problem detection. In IEEE International Workshop on Systems Management, Newport, Rhode Island. IEEE.

[18] Thottan, Marina and Ji, Chuanyi. (1998) Proactive anomaly detection using distributed intelligent agents. IEEE Network, Special Issue on Network Management.

[19] Williams, R. J. and Zipser, D. (1989) “Experimental analysis of the real time recurrent learning algorithm," Connection Science, 1, pp. 87-111

FIGURES

Figure 1: (a) left - example of a data communication network. (b) right - a neural network representing the connectivity of the data communications network (a).

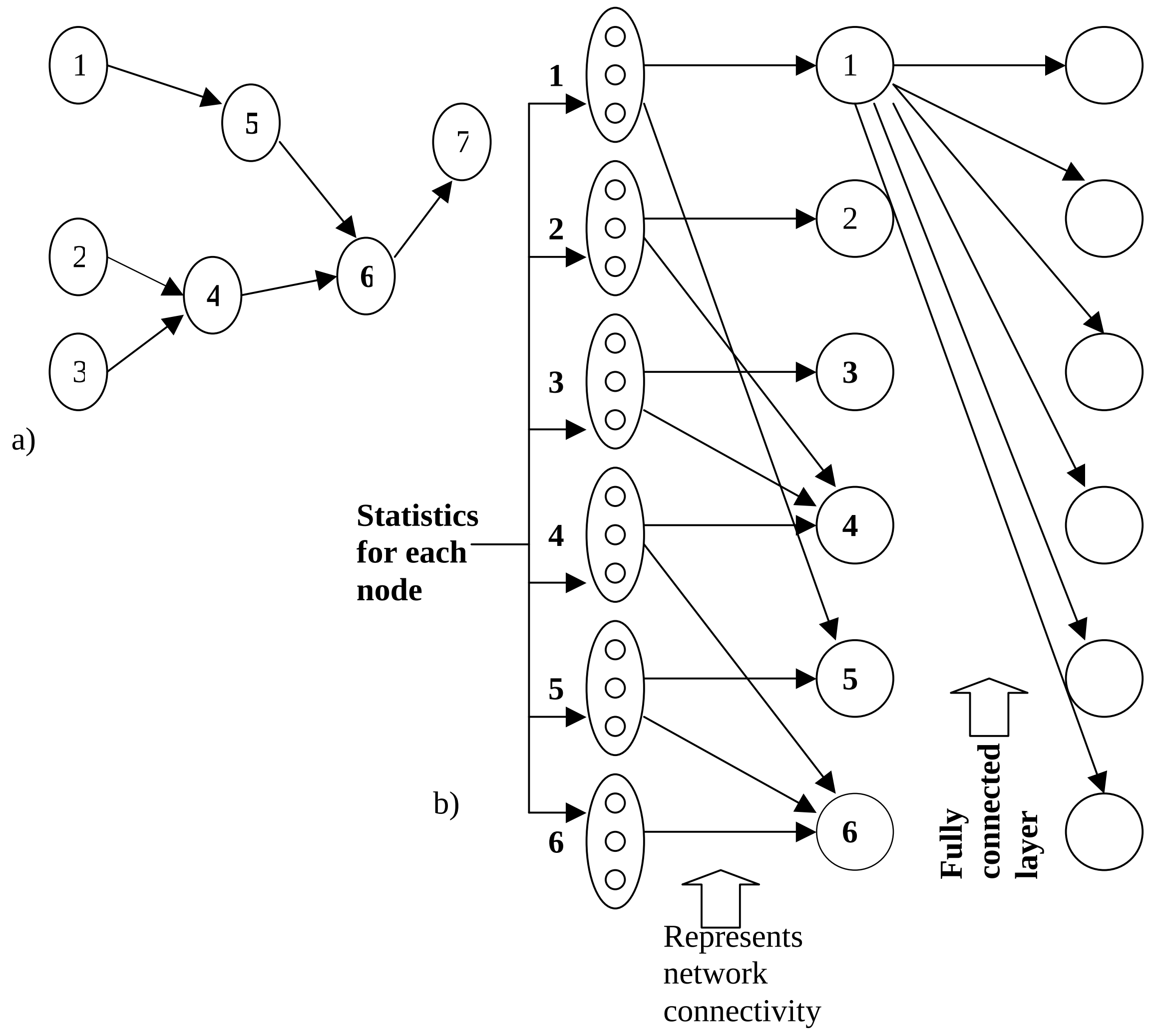


Figure 1: This figure depicts the relationship between the data communication network (a) and the neural network (b). The neural network reflects the connectivity of the data network by only allowing links between neurons representing adjacent nodes. For example, statistics from node 1 are communicated to the hidden layer nodes 1 and 5. The connection between node 1 in layer 1 and node 1 in the hidden layer exist because in the data network, they represent the same node. However, the connection between node 1 in layer 1 and node 5 in the hidden layer exist because in the data network, 1 and 5 are adjacent nodes.

Figure 2: Control Agent Architecture

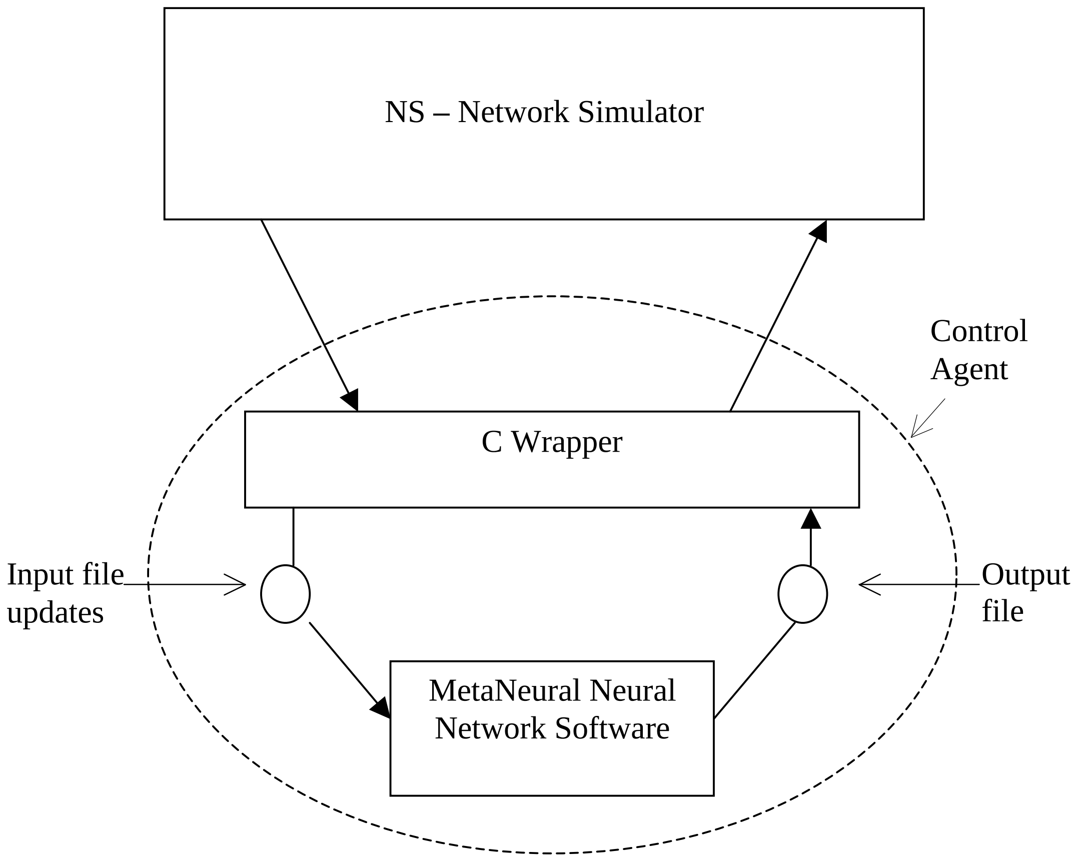
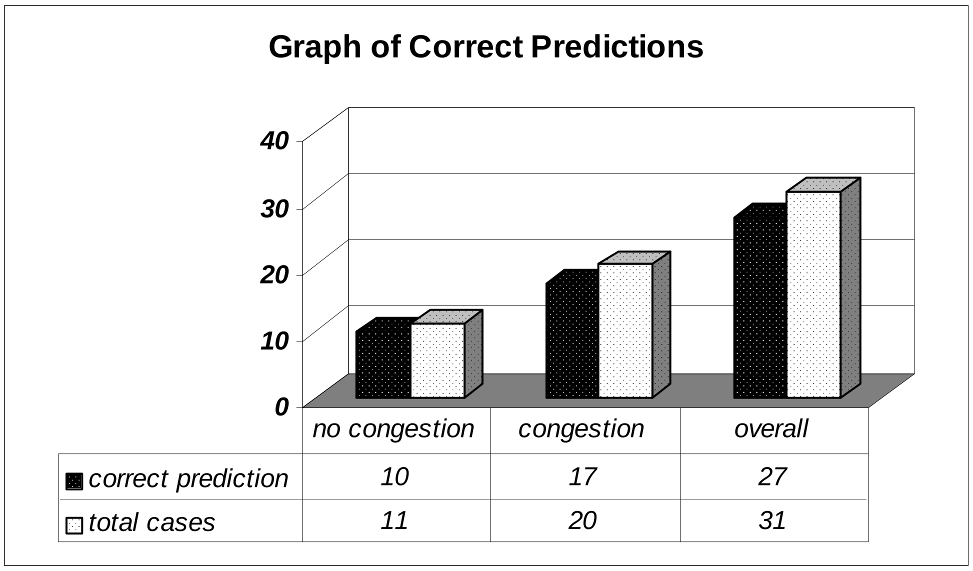


Figure 2: This Figure shows the relationship among the different components of our system. It also depicts the flow of information among these components. All components inside of the dashed circle are considered to be part of the Control Agent.

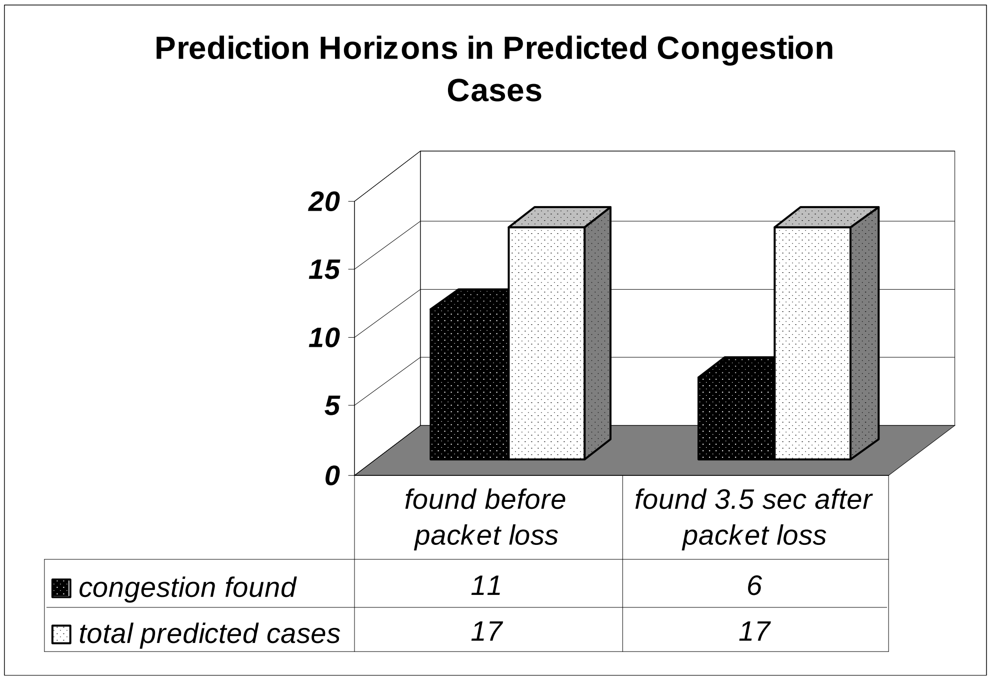
Figure 3: Breakdown of Results



	no congestion	congestion	overall
■ correct prediction	10	17	27
□ total cases	11	20	31

Figure 3: This is the breakdown of the system results over the different cases of congestion. This graph shows that the system was able to successfully detect congestion (or detect no congestion) in about 90% of the test cases.

Figure 4: Prediction Horizon Graph



	found before packet loss	found 3.5 sec after packet loss
▩ congestion found	11	6
▢ total predicted cases	17	17

Figure 4: This graph shows the amount of time it took our system to determine that congestion was present. Note that over 60% of the cases were determined to cause congestion before any packet loss had occurred.

Figure 5: Extension of the Architecture

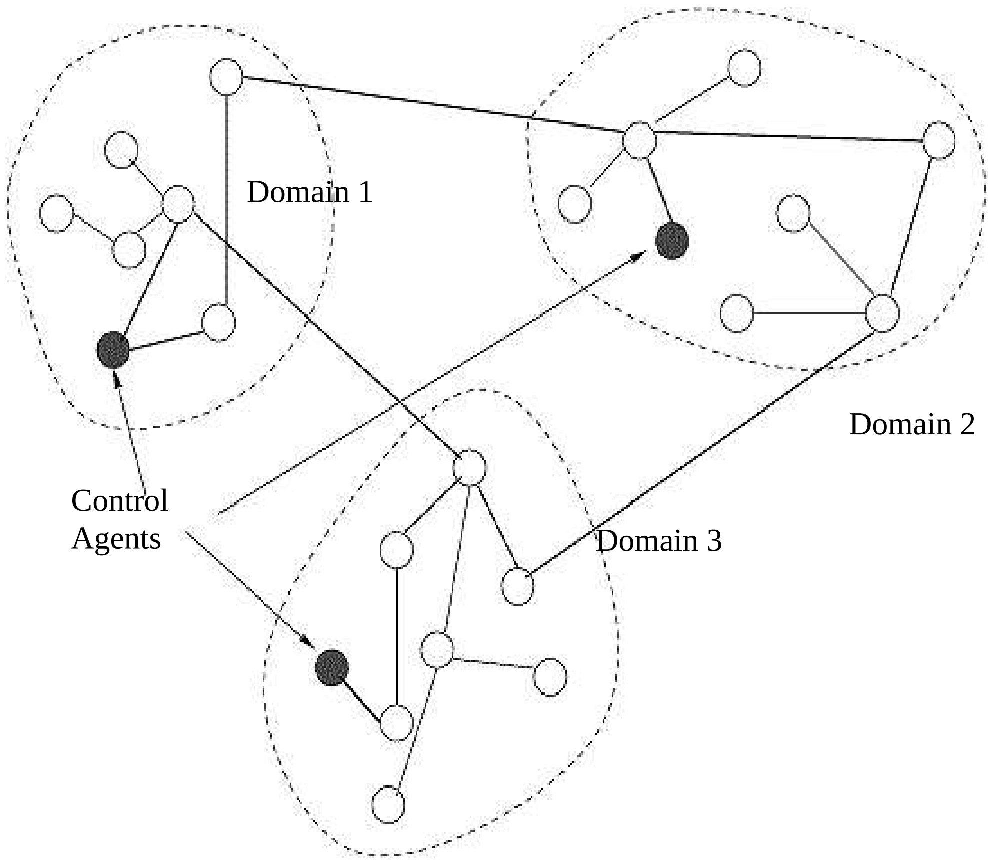


Figure 5: This figure shows a possible extension of the architecture for managing large networks. As pictured, it involves splitting these large networks into smaller networks. Each smaller network is managed independently with boundary traffic handled by a separate protocol between Control Agents.