Appeared in: Proceedings of the IEEE MILCOM, Atlantic City, NY, Nov. 1999

Preventing Traffic Analysis for Real-Time Communication Networks

Yong Guan, Chengzhi Li, Dong Xuan, Riccardo Bettati, Wei Zhao
Department of Computer Science,
Texas A&M University
College Station, TX 77843-3112
{yguan, chengzhi, dxuan, bettati, zhao}@cs.tamu.edu

Abstract-In this paper, we address issues related to preventing traffic analysis in computer networks used for real-time mission-critical applications. We consider an IP-based network where headers of packets, including source host address and destination host address, are readable by an observer (i.e., by a potential enemy). Although the encryption of network packets significantly increases privacy, the density of the traffic can still provide useful information to the observer. We take an approach by manipulating traffic in the network through host-based rerouting and traffic padding so that the traffic shows a time-invariant pattern. Thus, the observer can not derive any useful information about the real traffic pattern. By evaluating the performance of the algorithms used for this problem in terms of acceptance rate and execution time, we found that some well-known theoretical optimal and near-optimal algorithms failed to meet one or the other criteria. In this paper, we present a heuristic method that can effectively prevent traffic analysis while at the same time meeting real-time requirements. Our algorithm generates a plan that specifies where and when the dummy packets should be transmitted and if and how the payload packets should be rerouted and can yield high acceptance rate with low execution time. The success of the algorithm stems from the fact that it explicitly takes into account of real-time requirements and properly balances the traffic over the links.

I. INTRODUCTION

In this paper, we address issues related to preventing traffic analysis in computer networks used for real-time mission-critical application. Prevention of traffic analysis has been one key issue in network security, especially for networks in the battlefield. Generally speaking, the traffic pattern within the network exhibits different characteristics under different situations and at different times. For example, the pattern of the traffic entering into and exiting from a military command control center is very different depending on the state of alertness at which the center operates. Although the encryption of network packets ensures privacy of the payload, the density of the traffic (e.g., the number and size of the messages entering into or exiting from a given site or region) can still give some useful information to an observer. Simply by observing the traffic pattern, the observer may deduce some important information and infer the current activity or intention of the network users. Information about traffic density can be easily obtained in wireless environments by observing the radio frequency sources. In wired networks, this can be done by properly placing packet sniffers, by using commercial network management tools, or by other means.

This problem can be simply stated as: "By the act of communicating, even if perfect confidentiality of the actual information is achieved, one can give indications to observing parties of impending actions, capabilities, chains of command, and level of readiness." (from "Research Challenges in High Confidence Networking", DARPA, July 1, 1998).

In addition to requiring a high degree of network security, many mission-critical applications in the battlefield have the real-time requirements as well: messages in the battlefield must be delivered within previously defined deadlines in order to be of value.

In this paper, we present a method that can effectively prevent traffic analysis while at the same time meeting real-time requirements of messages in the network.

We consider an IP-based network where headers of the packets, including source host and destination host addresses, are readable by an observer. Thus, the traffic pattern is defined as the amount of traffic that is transmitted between any two hosts. To prevent traffic analysis, we manipulate the traffic in the network so that the traffic shows a time-invariant pattern (we call this a *stable pattern*). That is, after manipulation, the pattern of the traffic in the network will appear the same *all the time and at any time*, regardless of how the real (payload) traffic in the network changes under different situations. In this way, the observer can not derive any information about the real traffic.

We achieve prevention of traffic analysis by two means: host-based rerouting and traffic padding. Traffic padding means that dummy packets are injected into the network in order to make the traffic pattern time-invariant. With host-based rerouting, a packet may be sent to some intermediate hosts first and then sent to its true destination. Through host-based rerouting, the true source and destination of the traffic can be hidden using appropriate encryption. Although the observer can obtain the current source host address and the current destination host address of the traffic that passes through a link, she may or may not be observing the true source and destination of the traffic. This can considerably complicate the task of traffic analysis. Host-based rerouting can also reduce the load on some links. Through host-based rerouting and traffic padding, the amount and nature of traffic between the source and destination within the network can be masked.

Once payload traffic specification, real-time deadline requirements, and stable traffic pattern constraints are given, we need to determine if there exists a plan that specifies where and when the dummy packets should be transmitted

and if and how the payload packets should be rerouted via some intermediate hosts. It is a hard problem to efficiently determine if the payload traffic specification is feasible under given stable traffic pattern constraints and real-time deadline requirements, i.e., if it is possible to find a plan that satisfies the payload traffic specification and real-time requirements and obeys the stable traffic constraints. In this paper, we report on an approach to address this problem.

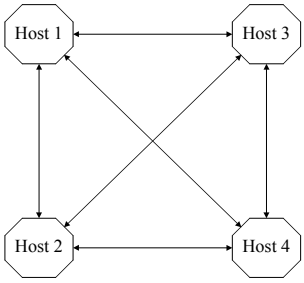

Figure 1. Fully Connected Directed Network

The rest of the paper is organized as follows: In Section 2, we present a brief survey of related work on prevention of traffic analysis, delay analysis and on mathematical methods used later in this paper. Section 3 describes the basic model and notation used in this paper. The definition of the problem is presented in Section 4. The delay analysis is presented in Section 5. The design of our heuristic algorithm used for prevention of traffic analysis in real-time communication networks is presented in Section 6. The performance evaluation is presented in Section 7. Section 8 concludes this paper with a summary of main results and suggestions for future extensions to this work.

II. RELATED WORK

A model to prevent traffic analysis by converting a given traffic matrix into a neutral traffic matrix is described [1]. The performance analysis of this model is given in [2][3]. This work has a limitation that the rerouted traffic can only be transmitted through one hop (one intermediate node between the source and destination). If we relax this limitation, we can obtain the same results at less cost. Some work about the delay analysis for real-time communication services over ATM and FDDI network had been done by our group in [4][5][12], and by many others. Given the stable traffic matrix, we are able to calculate the worst case delay of packets. Whether a real demand traffic matrix is feasible under the stable traffic matrix is a multi-commodity concurrent flow problem in nature. Much work on this problem is described in [6][7][8][9][10][11].

III. BASIC MODEL AND NOTATION

We assume that an underlying routing subsystem determines a unique path between any pairs of hosts. For our purposes, we model the network at transport layer and assume that every host can communicate with every host. The network therefore can be modeled as a fully connected, directed graph. All the edges in the fully connected directed graph represent the paths from the source host to destination host through some routers in the network.

An observer can read/monitor the amount of traffic along each edge of the graph. Thus, this observer can determine the traffic pattern between each source-destination pair. Our work will be aimed at preventing such information from being released to the enemy observer.

Definition 1: The *Real Demand Traffic Matrix* $A=(a_{ij})_{nxn}$ denotes the actual bandwidth requirement between any two hosts. The values a_{ij} are constraint so that the row-sum of any row i does not exceed the capacity of the output link of Host i. Similarly, the column-sum of any column j does not exceed the capacity of the input link of Host j.

In the following we will use the matrix A represented below as example traffic matrix for the system of four nodes illustrated in Figure 1.

$$A = \begin{pmatrix} 0 & 0 & 3 & 3 \\ 3 & 0 & 3 & 3 \\ 2 & 5 & 0 & 2 \\ 3 & 3 & 3 & 0 \end{pmatrix} \quad (5)$$

Definition 2: The *Stable Traffic Matrix* $B=(b_{ij})_{nxn}$ represents the bandwidth requirement of the stable traffic pattern between any two hosts. This can be monitored or detected by the observers. Again, row and column sums are constraint as to not to exceed host output or input link capacities.

Definition 3: Given a network $G = (V, E)$ with vertex set V and edge set E, V contains all the source and destination of the traffic flow. Edge uv in E represents the actual traffic flow from Node u to Node v, which may contain the direct traffic from Node u to Node v or any rerouted traffic that uses Node u or Node v as intermediate nodes. The *traffic flow f* consists of n^2 vectors f_{ij}, where $f_{ij}(u,v)$ represents the amount of traffic from Node i to Node j on edge uv.

Definition 4: The *Padding Traffic Matrix* $C=(c_{ij})_{nxn}$ represents the amount of the padding traffic from any Host i to any Host j. The following two constraints hold for C:

$$0 \le c_{ij} \le b_{ij} \quad (11)$$

$$\sum_{1 \le u,v \le n} f_{uv}(ij) + c_{ij} = b_{ij}, \quad (12)$$

Definition 5: The *Worst-Case Delay Matrix* $D=(d_{ij})_{nxn}$ represents the worst-case message delivery delay for the traffic flow from any Host i to any Host j.

Definition 6: The *Deadline Matrix* $DL=(dl_{ij})_{nxn}$ represents the deadline requirements for the traffic flow between any two hosts.

Definition 7: Rerouting Quantity Column vector: R

According to the rerouting path length (the number of hops, i.e., the number of intermediate nodes in the path), we have the following rerouting quantity column vectors: R_1, R_2,

... , R_{n-2}. For example, if the rerouting path length is k, R_k will be a n^{k+2} column vector whose elements are all $r_{im(1)...m(k)j}$ arranged in lexicographic order. Here $r_{im(1)...m(k)j}$ represents the amount of the rerouting traffic passing along the path $i \rightarrow m_1 \rightarrow ... \rightarrow m_k \rightarrow j$, where the true source is Host i and the true destination is Host j. And

$$f_{ij}(u,v) = \sum_{\substack{uv \in P \\ P=i...j}} r_P \qquad (17)$$

Definition 8: The *Payload Traffic Specification* $S=\{M_i\}$ is the set of all trafic flow requirements. Each M_i represents the bandwidth and real-time deadline requirement of the the i^{th} traffic flow in the system. Each Payload Traffic Specification corresponds to a real demand traffic matrix A and a deadline matrix DL.

Suppose that we choose the stable traffic matrix to be

$$B = \begin{pmatrix} 0 & 3 & 3 & 3 \\ 3 & 0 & 3 & 3 \\ 3 & 3 & 0 & 3 \\ 3 & 3 & 3 & 0 \end{pmatrix} \qquad (19)$$

This means the traffic pattern between any two hosts stays constantly at 3 MB/sec, and the observer can only know this traffic pattern among the hosts in the network. Through our approach, we can efficiently determine whether a real demand traffic matrix can be converted into the stable traffic matrix by host-based rerouting and traffic padding. This example is feasible, because we can have the following solution:

$$A = \begin{pmatrix} 0 & 0 & 3 & 3 \\ 3 & 0 & 3 & 3 \\ 2 & 5 & 0 & 2 \\ 3 & 2 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 3 & 3 \\ 3 & 0 & 3 & 3 \\ 2 & 3 & 0 & 2 \\ 3 & 2 & 3 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad (20)$$

Direct Host-based Rerouting Padding

Through this manipulation, the amount (5 MB/sec) of real traffic from Host 3 to Host 2 is divided into the following three parts: 3 MB/sec of the traffic from Host 3 to Host 2 are sent directly. 1 MB/sec of the traffic is rerouted to Host 1 first and then sent to Host 2, and 1 MB/sec of the traffic is rerouted to Host 4 first and then sent to Host 2. 2 MB/sec of the padding traffic are injected into the traffic from Host 1 to Host 2 such that the stable traffic pattern can be achieved. After this manipulation, the traffic between each source-destination pair remains to be 3 MB/sec all the time. This is the stable traffic pattern that the observers can obtain. From this, the observer can not find any information about the real traffic pattern.

IV. Definition of the Problem

Consider a network with a high degree of security and real-time requirements. We model this network as n nodes, which form a fully connected directed graph. The network operates at any one of m different modes, which correspond to m different known states or situations of this system in the real world. Each mode reflects one state or one situation of the system, which contains a set of traffic flows in the network system. Thus each mode corresponds to a real demand traffic matrix A and a deadline matrix DL.

We define the value of the total traffic flow on edge uv to be $f(uv) = \sum_{ij} f_{ij}(uv)$. The real demand traffic matrix A is feasible, if (1) $f(uv) \le b_{uv}$ for all edges uv, where b_{uv} is an element of the stable traffic matrix B, and (2) $d_{ij}^{WC} \le DL_{ij}$ for all the traffic flows in the real demand traffic matrix. d_{ij}^{WC} is the worst-case message delivery delay for the real traffic flow from Host i to Host j. If a real demand traffic matrix is feasible, the traffic flow $f_{ij}(uv)$ can be generated, which is a plan that specifies where and when the dummy packets should be transmitted and if and how the payload packets should be rerouted.

In this paper, we will focus on determining if the real demand traffic matrix is feasible, i.e., if it can be converted into the stable traffic matrix through host-based rerouting and traffic padding and the real-time requirement can also be satisfied at the same time.

A correct solution for this problem must satisfy the following four sets of constraints:

A. Stabilization Traffic Constraints:

We must ensure that the real demand traffic can be sent directly or rerouted through some intermediate node(s) and the final traffic pattern is stable. Formally,

$$\sum_{1 \le u,v \le n} f_{uv}(ij) \le b_{ij} . \qquad (22)$$

B. Link Capacity Constraints (i.e., Bandwidth Constraints)

Constraints on row- and column-sums of the stable traffic matrix make sure that no bandwidth capacities are exceeded, that is,

$$\sum_{j=1}^{n} b_{ij} \le \text{the capacity of the output link from host i.} \qquad (23)$$

$$\sum_{i=1}^{n} b_{ij} \le \text{the capacity of the input link into host j.} \qquad (24)$$

C. Conservation Constraints:

These ensure that the correct amount of traffic is rerouted at each node. If a node is the source or destination of the traffic, the aggregate output or input traffic must be equal to the outgoing or incomming real demand traffic, respectively. Formally,

$$\sum_{vu \in E} f_{ij}(vu) = a_{ij} \quad \text{for } v=i \text{ and Host } i \text{ is source} \qquad (26)$$

$$\sum_{uv \in E} f_{ij}(uv) = a_{ij} \quad \text{for } v=j \text{ and Host } j \text{ is destination} \qquad (27)$$

If the node is used as intermediate node in some rerouting path, the aggregate input rerouting traffic of this node must be equal to its aggregate output rerouting traffic.

$$\sum_{uv \in E} f_{ij}(uv) - \sum_{vu \in E} f_{ij}(vu) = 0 \text{ for } v \notin \{i,j\}. \quad (25)$$

D. Delay Constraints

In order to guarantee the real-time requirements we must ensure that all traffic can be sent to their destination by its deadline, Formally,

$$d_{ij}^{WC} \le DL_{ij} , \quad (28)$$

where d_{ij}^{WC} is the worst-case message delay for the traffic flow from Host *i* to Host *j*. This will be discussed in the following section.

V. Delay analysis

Determining end-to-end delays in communication networks has been the subject of a large amount of research. The delay computation in this case is somewhat complicated by the fact that we do host-based redirection.

In order to determine the end-to-end delay of trafic flows, we distinguish between the *direct path* of the flow, which is the path from the source host to the destination host as determined by the underlying routing subsystem (e.g., OSPF,) and the *rerouted path*,which is the path assigned to the flow by the rerouting subsystem. We denote the worst-case delays of messages along the two paths by $d_{ij}^{WC\_direct}$ and, respectively. Once the delay along the direct paths within the network are known, the value for $d_{ij}^{WC\_reroute}$ is determined by taking the maximum of the delays along all the host-based rerouting paths used to transmit the traffic between the two hosts. The delay along each rerouting path is computed by summing up the delays on the direct paths between host-based routers on the rerouting path. The end-to-end worst case delay d_{ij}^{WC} of a traffic flow is then formulated as:

$$d_{ij}^{WC} = \max\left(d_{ij}^{WC\_direct}, d_{ij}^{WC\_reroute}\right), \quad (29)$$

The worst case direct delay experienced by a packet of traffic flow M_i is computed by summing the local delays at each server traversed along the direct path of the connection.

The scheduling policy at a server determines the order in which packets from a traffic flow are transmitted at the output of the server. Hence, the server scheduling policy has a direct impact on the delays experienced by a traffic flow's packet at a server as well as on the distortion of the traffic flow's traffic within the network.

Formulas for the delay at servers for a variety of servers exist. For First-Come-First-Served (FCFS) servers, the worst case delay experienced by any packet at the server is the same for any traffic flow traversing it. For an network with FCFS-based servers, the maximum delay at Server *j* is given by

$$d_{*,j} = \max_{I>=0}\left(\sum_{k=1}^{L_j} \hat{F}_{k,j}(I) - I\right), \quad (34)$$

where $\hat{F}_{k,j}(I)$ is the maximum number of packets that can arrive at Server *j* over its k^{th} input link during any interval of length *I*, and L_j is the number of input links into Server *j*.

Similarly, in [12] we derive a delay formula for networks with static priority schedulers with a fixed, globally distinct, priority assignment (SFGDP). A priority assignment is fixed if the priority of the packets in a traffic flow is the same in different routers along the host-to-host path. And the priority assigned to each traffic flow is globally distinct, i.e., none of the traffic flow's priority have the same priority. A wealth of other delay formulas for other scheduling policies in the servers is available in the literature.

VI. Algorithms

A. Overview

Once the payload traffic specification, the real-time deadline requirements and the stable traffic pattern constraints are given, we want to determine whether the real demand traffic matrix is feasible as described in Section IV under the stable traffic matrix. And, if feasible, a schedule needs to be generated that specifies where and when the dummy packets should be transmitted and if and how the payload packets should be rerouted through intermediate hosts.

The design of an algorithm that determines the feasibility of real demand traffic matrix under the stable traffic matrix must have two primary objectives:

- High Acceptance Rate: The ratio of the number of sets of traffic requirements that can be admitted by the algorithm over the total number of sets of traffic requirements must be high for the algorithm to be effective.
- Low Execution Time: The average time that the algorithm needs to process one set of payload traffic requirements must be low for the algorithm to be efficient.

Effectiveness and low computation costs must be traded off against each other: In order to obtain a higher acceptance rate, the algorithm always needs a longer execution time. On the other hand, the algorithm often gets a lower acceptance rate if a shorter execution time is required.

We have identified three solution approaches, depending on whether the resulting algorithm should be optimal, near-optimal, or heuristic. We use a Linear Programming formulation of the problem to derive an optimal algorithm. We then proceed to develop a near-optimal algorithm, which is derived from a multi-commodity flow formulation of the problem.

We complete this algorithms suite with a heuristic algorithm for this problem that yields high acceptance rate and has low execution time at the same time.

In the following sections, we elaborate on each of the proposed three algorithms in turn.

B. Optimal Algorithm: Linear Programming

Linear Programming is a theoretical optimal algorithm for this problem. Consider a network with n hosts, for one source-destination pair, we have a total of $\sum_{k=0}^{n-2} \frac{(n-2)!}{(n-2-k)!}$ possible host-based rerouting paths. To formulate an instance of this problem defined in Section IV into an instance of the linear programming problem, we need to introduce $\sum_{k=0}^{n-2} \frac{n!}{(n-2-k)!}$ unknown variables $f_{ij(p)}$, $1 \le i, j \le n$, where $f_{ij(p)}$ represents the amount of the traffic flow from Host i to Host j through the host-based rerouting path p. The constraints can be easily formulated into the following linear relations,

For $1 \le i,j \le n$, $\sum_{p \in P} f_{ij(p)} = a_{ij}$, (36)

where P is the set of all the possible host-based rerouting paths whose worst-case delay is less than or equal to the deadline. a_{ij} is the bandwidth of the traffic flow from Host i to Host j.

If $uv \in p$, i.e., p is a host-based rerouting path that contains edge uv, we have

$$0 \le \sum_{p \in Q} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij_p} \le b_{uv}, \quad (37)$$

Where $Q = \{q \mid uv \in q$, and q is a host-based rerouting path$\}$, and b_{uv} is the time-invariant traffic pattern.

For $1 \le i,j \le n$, $0 \le f_{ij(p)} \le a_{ij}$. (38)

Since we only need to find a solution that satisfies the bandwidth requirements, stable traffic pattern constraints and real-time deadline requirements, the objective function can be zero. Therefore, the instance of this problem has been formulated as an instance of the Linear Programming Problem.

However, linear programming is not a practical method. The very quickly growing number of variables makes it very difficult for current mathematical software, like IMSL [14] and NAG [15], to solve non-trivial networks. For a network of size 8, for example, the total number of variables in the linear programming formula is 109,592.

C. Near-optimal Algorithm: Multi-commodity Flow

The concurrent multi-commodity flow problem involves simultaneously shipping several different commodities from their respective sources to their destinations in a single network so that the total amount of flow going through each edge is no more than its capacity. Associated with each commodity is a demand, which is the amount of that commodity that we wish to ship. In this paper, we aim to prevent traffic analysis by making the traffic pattern time-invariant. So the amount of the traffic between any two hosts can not exceed the time-invariant traffic pattern constraints specified in the given stable traffic matrix. We can use the fully connected directed network to represent this model. The element in the given stable traffic matrix is regarded as the capacity of the edge in the network. Each payload traffic flow in the real demand traffic matrix is regarded as a commodity. The bandwidth requirement of the traffic flow is the demand of the commodity. The real-time deadline requirement of the traffic flow limit the choice of the host-based rerouting path, i.e., we can only choose those host-based rerouting path whose worst case delay is less than or equal to its deadline. Therefore, the instance of this problem has been formulated as an instance of the concurrent multi-commodity flow problem.

Although this problem is *NP*-complete, it can quite well be solved approximately. Algorithm LMPSTT [6], for example, is a well-known fast approximation algorithm for this problem. It is known to be ε-optimal: given a multi-commodity flow problem, Algorithm LMPSTT can answer if it is feasible, and if feasible, give a feasible flow for the problem in which the capacity on every arc is increased by a factor $(1+\varepsilon)$.

By taking the real-time requirements of the payload traffic into consideration, we obtain Algorithm Revised-LMPSTT that can be used to solve our problem and meet the real-time deadline requirements. Since the network we consider here is a fully connected directed graph, we will use the element in the stable traffic matrix as the capacity of the corresponding edge in the network, because the amount of the total traffic on an edge can not exceed the stable traffic pattern constraints. Each payload traffic flow in the real demand traffic matrix is regarded as a commodity. Algorithm Revised-LMPSTT will generate a plan that specifies where and when the dummy packets should be transmitted and if and how the payload packets should be rerouted if the problem is feasible under the stable traffic pattern constraints and the real-time deadline requirements.

Although Algorithm Revised-LMPSTT is a near-optimal algorithm and may achieve a high acceptance rate, its computation cost is high. This is partly due to the need to take into consideration deadline requirements. Moreover, Algorithm Revised-LMPSTT needs a large number of iterations for a small value of ε.

D. Heuristic Algorithms

Figure 2 shows a simple framework for heuristic algorithms to solve our problem. The algorithms repeatedly select a flow to schedule, find a path for the flow, adjust, the available link capacities, and either terminate successfully if all flows have been scheduled, or abort if a flow cannot be routed.

Two decision steps affect the performance of the heuristic algorithm; and a variety of algorithms can be devised based on how these steps are implemented in detail: (1) selection of the next traffic flow to be scheduled in Step 2 and (2) selection of the path to be used to transmit the selected traffic flow in Step 4. By varying the selection of traffic flows and paths, the following heuristic algorithms can be defined:

Algorithm Type-1: Picks the next traffic flow at random, and picks the next path with the maximum capacity.

Algorithm Type-2: Picks the next traffic flow at random, and picks the next path with the minimum delay.

Algorithm Type-3: Considers the real-time requirement of the traffic flow, and the demand bandwidth of the payload traffic and the available bandwidth in the network, and picks the path with the maximum capacity.

Algorithm Type-4: Same as Type-3, but picks the path with the minimum delay in the network.

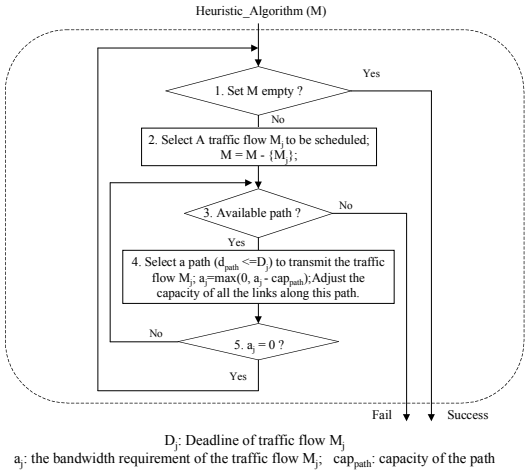

Figure 2. Framework for Heuristic Algorithm

Although the heuristic algorithm is not optimal, but Algorithm Type-4 can yield high acceptance rate at low execution time. We demonstrate this in the performance evaluation experiment described below.

VII. PERFORMANCE EVALUATION

In this section, we will report performance results of the algorithms we discuss in this paper.

We consider a simple network with 5 nodes, and we assume that all nodes communicate with all other nodes. We designed a generator to generate 1000 random sets of real demand Matrix *A*, Deadline Matrix *DL*, Delay Matrix *D*, and Stable traffic Matrix *B*. To precisely determine the performance of the algorithms, we put some constraints on this generator such that each set of generated matrices is feasible. In our simulation, we use the value of α to represent the system load factor, which represents the system utilization. We also use the value of ε for the approximation algorithm to find an ε-optimal solution for this problem.

In these experiments, we compare the optimal algorithm (Algorithm LP), the near-optimal algorithm (Algorithm Revised-LMPSTT), and the various heuristic algorithms (Algorithm Type-1, Type-2, Type-3, and Type-4). As performance measures for comparison, we are interested in the acceptance rate and the execution time of each algorithm.

In order to obtain the performance data, we explicitly programmed Algorithm Revised-LMPSTT, and the four heuristic algorithms. We used a library package [15] for Algorithm LP. All experiments were performed on a SGIPower Challenge 10000 XL machine.

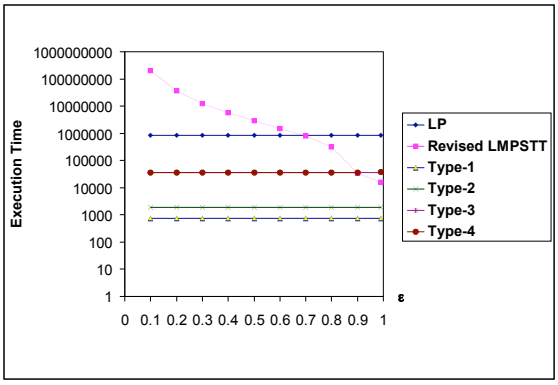

Figure 3. Execution Time vs. Accuracy.

Figure 3 shows the actual execution time for processing one real demand traffic matrix for varying values of ε. As the value of ε has no effect on the behavior of Algorithm LP and of the four heuristic algorithms, their execution times do not change as the value of ε changes. The execution time of Algorithm Revised-LMPSTT increases greatly as the value of ε decreases.

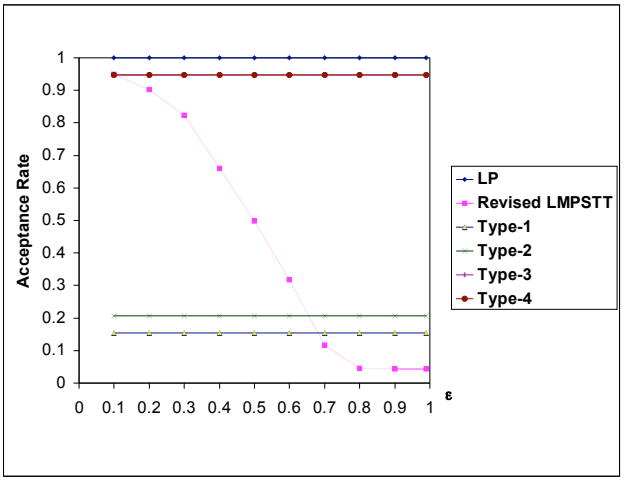

Figure 4. Acceptance Rate vs. Accuracy.

Figure 4 compares the acceptance rate of all algorithms for different value of ε. Throughout this experiment, we fix the load factor α at 0.98. The acceptance rate of Algorithm Revised-LMPSTT increases as the value of ε decreases, and almost reaches 90% when the value of ε is 0.2. The acceptance rate of Algorithm Type-1 and Algorithm Type-2 are very low. The result was to be expected, as the two heuristics do not attempt to make an informed selection of the next traffic flow.

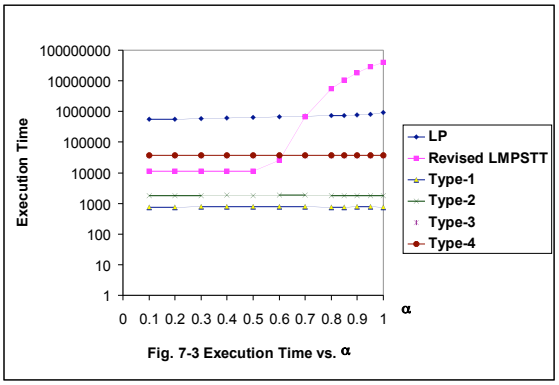

Figure 5. Execution Time vs. System Load.

shows the actual execution time for processing one real demand traffic matrix for varying degrees of system load (α). Throughout this experiment, we fix the value of the value of ε to 0.2. We observe that the execution times of both Algorithm Revised-LMPSTT and Algorithm LP increase as the value of α increases. However, the execution time of Algorithm Revised-LMPSTT increases much faster than that of Algorithm LP.

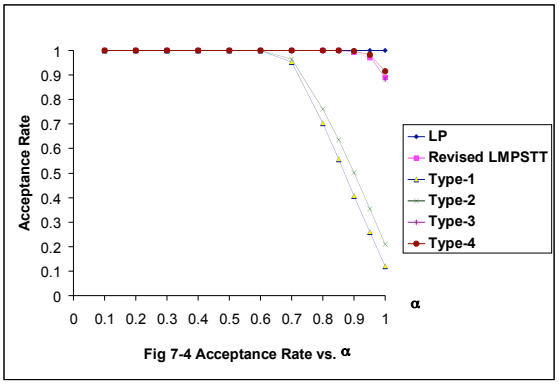

Figure 6. Acceptance Rate vs. System Load.

Figure 6 illustrates the effect of the system load on the acceptance rate of the algorithms. In this experiment, the value of ε is 0.2. We observe that the acceptance rates of Algorithm Type-1 and Algorithm Type-2 decrease very fast when the value of α is larger than 0.7 and is very low when value of α is 1.0. The acceptance rate of Algorithm Type-4 is bigger than that of Algorithm Type-3 and Algorithm Revised-LMPSTT when the value of α is 1.0, due to the large value for ε.

Generally, we observe that, in order to obtain the same reasonable acceptance rate, the execution time of Revised LMPSTT algorithm is greatly larger than that of all the four heuristic algorithms.

VIII. Conclusion and Future Work

In the context of this problem, Algorithm Type-1, Algorithm Type-2, Algorithm Type-3 and Algorithm Type-4 are heuristic, non-optimal, algorithms. Algorithm Revised LMPSTT is an approximation algorithm, and Algorithm LP is optimal.

However, Algorithm LP has shown to be impractical due to overly complicated constraints and exceedingly many variables needed for the mathematical model even for a small network with a few nodes. This makes the approach unusable for networks with large numbers of nodes. Algorithm Revised-LMPSTT is a polynomial-time combinatorial algorithm for approximately solving this problem. However, Algorithm Revised-LMPSTT is still not efficient enough, in particular when compared to the proposed heuristic algorithms, as it still needs a very long time to compute a solution. Among the proposed heuristic algorithms, Algorithm Type-1 and Algorithm Type-2 have a disadvantage that the acceptance rate is too low. Although Algorithm Type-3 almost has the same execution time as Algorithm Type-4, the acceptance rate of Algorithm Type-4 is higher than that of Algorithm Type-3. So we conclude that Algorithm Type-4 is a good algorithm for its high acceptance rate and low execution time compared to other algorithms.

The following are several possible extensions of the future work related to this problem.

Optimal stable traffic matrix. Given *m* modes, an optimal stable traffic matrix need to be determined. A stable traffic matrix is optimal if it minimizes the usage of bandwidth and the real demand traffic matrices for all operation modes are feasible under the optimal stable traffic matrix.

Restricted set of intermediate nodes. We are assuming a simplistic network model that allows direct communication between any two nodes in the network. In reality, this may not be the case. In addition, we assume that all nodes can forward (reroute) traffic on behalf of other nodes. Various security or other policy reasons may inhibit the free choice intermediate nodes.

Openness. We are currently addressing how to implement the work of prevention of traffic analysis in an open environment. Here openness means that the several private networks are connected through some public network, like the Internet. We need to do some in-depth research work on how to prevent traffic analysis in this case.

References

[1] R. E. Newman-Wolfe, B. R. Venkatraman, "High Level Prevention of Traffic Analysis," Seventh Annual Computer Security and Applications Conference, San Antonio, Texas, Dec 2-6, 1991

[2] R. E. Newman-Wolfe, B. R. Venkatraman, "Performance Analysis of a Method for High Level Prevention of Traffic Analysis," Eighth Annual Computer Security and Applications Conference, San Antonio, Texas, Nov 30-Dec 4, 1992

[3] B. R. Venkatraman, R. E. Newman-Wolfe, "Performance Analysis of a Method for High Level Prevention of Traffic Analysis Using Measurements from a Campus Network,"

Tenth Annual Computer Security and Applications Conference, Orlando, Florida, Dec 5-9, 1994
[4] A. Raha, S. Kamat, W. Zhao, "Guaranteeing End-to-End Deadlines in ATM Networks," Proc. Of International Conference on Distributed Computing System, 1995
[5] B. Devalla, A. Sahoo, Y. Guan, C. Li, R. Bettati and W. Zhao, "Adaptive Connection Admission Control for Mission Critical Real-Time Communication Networks," Proceedings of MilCom'98
[6] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas, "Fast approximation algorithms for multicommodity flow problems," Journal of Computer and System Sciences, 50(2): 228-243, April 1995
[7] C. Stein, "Approximation Algorithms for Multicommodity Flow and Scheduling Problems," Ph.D. thesis, MIT, Cambridge, MA, August 1992
[8] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by successive approximation," Mathematics of Operations Research, Vol. 15, No. 3, August 1990
[9] R. K. Ahuj, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan, "Finding minimum cost flows by double scaling," Math. Programming, vol. 53, 1992
[10] S. Kapoor and P. M. Vaidya, "Fast algorithms for convex quadratic programming and multicommodity flows," Proceedings of 18th Annual ACM Symposium on Theory of Computing, 1986
[11] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," J. Assoc. Comput. Mach. 37, 1990
[12] C. Li, A. Raha and W. Zhao, "Stability in ATM Networks," Proceedings of IEEE INFOCOM, 1997.
[13] D. J. White, "Operational Research," John Wiley & Sons Ltd., 1985
[14] IMSL, "IMSL MATH/LIBRARY Fortran Subroutines for Mathematical Applications User's Manual," IMSL Inc., 1989
[15] NAG, "The NAG Fortran Library Introductory Guide, Mark 16," NAG Ltd., 1993