A Model for Window Based Flow Control in Packet-Switched Networks

Xiaowei Yang
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract— **Recently, networks have increased rapidly both in scale and speed. Problems related to the control and management are of increasing interest. The average throughput and end-to-end delay of a network flow are important design factors. However, there is no satisfactory tool to obtain such parameters. The traditional packet-by-packet event driven simulation is slow when the network speed is high. The time driven simulation faces the difficulty of choosing the right time interval when simulating packet-switched networks. As Transmission Control Protocol (TCP) is the most widely used transport layer protocol, and it uses window based flow control mechanism, classic queuing theories involving Markov Chain assumptions are not applicable.**

This paper describes a model for window based flow control packet-switched networks. The model attempts to provide a way to obtain the steady state results for large and high speed networks using TCP. In this paper, we discuss in detail the construction, implementation and application of the model. This paper also compares the results obtained from the model with those from the packet by packet event driven simulation. The comparison shows the model is correctly modeling the networks.

I. Introduction

The Internet has been growing rapidly in recent years, which results in problems related to routing, flow control, administration, etc. Simulation and analysis are the two commonly used methods to study the characteristics of networks. However, when networks become very large, neither method is easy to implement. For example, the traditional packet by packet level event driven simulation is slow when simulating a high speed network. The fixed granularity of the event driven simulation is its bottleneck. Unless we change the abstraction level, it is hard to improve its performance. The analytic method often uses classic queuing and network stochastic models which are restricted to problems that can be approximated as Markov chains [1]. As networks use complicated protocols such as TCP/IP to implement flow control, they can't be simply modeled by Markov chain assumption. Thus, we lack a satisfactory model to study the behavior of window based flow control packet-switched networks.

In this paper we present a simple and applicable model for window based flow control packet-switched networks. The model ignores the packet-level behavior of networks. Instead, it gives a higher level description, such as the average flow rate and the average queue length at each outgoing link. Incorporating this model into simulation methods will hopefully provide a practical way to study the behavior of large scale networks.

Section II is a review of the related work. Section III introduces the model. It includes basic assumptions the model depends on and it also describes the meaning of every formula in the model. Section IV presents an algorithm for finding a solution for the model. Section V discusses the implementation details. Section VII shows how to apply this model to simulate TCP with Random Early Detection(RED) [2]. The last section is a short summary of the paper and an assessment of future work.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under contract DABT63-94-C-0072.

II. Related Work

For rate based flow control networks, there are known solutions to find the average throughput of each flow. For example, the *max-min* fair rate allocation algorithm [1] computes each flow's rate for a given network's configuration and a set of flows with specified paths. Because the sending rate of each flow and the end-to-end delay are both unknown parameters in window based flow control networks, there seems no equivalent algorithm to compute each flow's average sending rate, end-to-end delay, nor the queuing delay at each router. In order to study large and complicated networks, more and more researchers are using simulation instead. There are two effective ways to simulate network. One is the discrete event driven simulation. The other is the discrete time driven simulation. Since in most networks, data are chopped into discrete packets, discrete event driven simulation is straightforward and it often can grasp the nature of networks.

The current academically popular network simulator, is *ns* [3], developed by the Network Research Group at Lawrence Berkeley National Laboratory(LBNL). *ns* is a discrete event driven simulator which can simulate a wide range of protocols, such as TCP and other routing, multicast protocols. The representation of an event in *ns* is the state of a packet. Typical events are the arrival or departure of a packet from a queue. Thus, detailed information of each packet's behavior can be obtained via simulation. *ns* has a global scheduler that manages an event queue. This queue is arranged in the time sequence that the events should happen. The simulation proceeds by simulating each event in the order they are stored in the queue. As one event takes place, it can trigger other events. These events are also inserted into the queue based on the time they are to happen. For the packet level event driven simulation, the simulator has to simulate each packet's behavior even if we are not interested in such a fine abstraction level. When the network's size is moderate, *ns* is an ideal tool to study network protocols. The detailed information about each packet's trace, such as when the packet arrives at a specific router, or when and where the packet is dropped, can be precisely recorded. However, discrete event driven simulation is slow when the speed of networks is high. First, the packet level granularity is fixed in spite of the size of

the network. Second, in order to update the effects caused by each event, the event order must be kept. A global event list has to be maintained, which results in an essentially sequential process. These two phenomena can not be avoided in the discrete event driven simulation.

A discrete time driven simulation updates the state of each simulated object at each delta time interval. Choosing a proper time interval is always a difficult problem. If the time interval is too large, the simulation will fail to detect events which should have occurred during the time interval. If such events are significant, the simulation's error bar will be high. On the contrary, as the simulator has to update the state of each simulated object at each delta interval, if the delta interval is too small, the simulator has to do updating very frequently. If we have a large network which contains a lot of objects and not every object changes its state frequently, the simulation is very ineffective. Especially when we are interested in a long time period simulation, it may also take a long time to do so many updates.

Another proposal is to use fluid model based time-driven simulation to simulate high speed networks [4]. This model simulates the network traffic as fluid. It is proved that the discretization error can be bounded by a constant proportional to the discretization time interval. This model can quickly locate the time and place of congestion. Besides, by the nature of time driven simulation, parallelism can be exploited to speed up the simulation. However, under what circumstance the network traffic can be modeled as fluid still needs further study. In the window flow control network, the rates of flows are unknowns. The fluid model needs the arrival rates of flows as input. We do not know whether the fluid model is adequate enough to model the window based flow control networks.

Also, there are many well developed queuing and stochastic network theories related to network flow and congestion control. When the network is large, those analytic models tend to contain some impractical assumptions such as Markov chain. Thus, it is not convenient to apply such queuing theories to window flow control networks.

In a word, for large scale and high speed window flow control networks, if we are interested in the long term average statistics such as rates and queuing, there is no satisfactory model. In this paper we introduce an analytic model for solving such problems.

III. Model Construction

This section describes the model for window based flow control networks.

A. Basic Assumptions

A flow of data between a sender A and a receiver B is said to be end-to-end window flow controlled if there is an upper bound on the data units that have been sent by A but are not known by A to have been received by B [1]. This upper bound is called the *window size*. In TCP, the upper bound is used both to keep the sender from overflowing the receiver's buffer and to keep the sender from overflowing the buffers inside the network. Window flow control can not guarantee a minimum sending rate nor a minimum packet delay. However, in some circumstances, we are still interested in knowing what average rates window-controlled flows could achieve and how large the queue size would be at each outgoing link if each flow's window size is known and fixed. We aim at a simple model which can provide us the steady state information fast and fairly accurately.

As we are interested only in the steady state, we want to give some assumptions to clarify the situation we are studying.

- Network configurations: assume we know the topology of the network, each link's capacity and propagation delay.
- Routing and Queuing Policy:

1. Assume fix route routing. All packets belonging to the same sender and receiver travel through the same path. The routing table of each router is known.
2. Assume routers keep a distinct output queue for each outgoing link. The processing delay of each packet at the router is negligible compared to the propagation delay, queuing delay and transmission delay.
3. The output buffer is infinite. We do not consider buffer overflow for now.
4. All data packets are of the same priority class. They are all arranged into a single queue and first come, first transmitted.

- Flows:

1. All flows have fixed window sizes, and the network is in a steady state, *i.e.*, each flow has a full window size of packets on the fly and the sender is sending at a steady state rate.
2. Senders always have data to send. In the period of our study, the set of flows is fixed.
3. There is only one-way traffic on each flow. When a receiver receives a data packet, it sends an ACK back immediately. ACKs travel through the same paths as those of the corresponding data packets[1]. Since ACKs are tiny packets, we assume ACKs are never backlogged at any link's output queue.

- Packets: All packets are 1000 bytes long and the length of ACKs can be ignored.

In the next section, we discuss our model based on the above assumptions.

B. The Model

We start from a simple example to get some intuition.

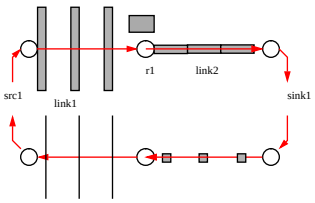

Fig. 1. A simple illustrative example

Figure 1 shows the example. There is only one sender src_1 which is sending with a window size W. By the network's steady state assumption, the flow has W packets on the fly. They are either flying inside the pipe, transmitted by src_1, queued or transimitted at r_1 or they have been received and are represented

[1] This assumption is not necessary. It only means to simplify programming. If ACKs travel through symmetric paths as those of data packets, only the forwarding paths of data packets need to be specified. The model is also applicable when ACKs travel through asymmetric paths. The only difference is we have to specify both data and ACKs' paths in order to calculate the round trip delay.

by ACKs returning from $sink_1$. Since the flow is sending at the steady state rate, the number of packets in the pipe is roughly $(\frac{t_1}{\frac{1}{\lambda}} + \frac{t_2}{\frac{1}{\lambda}})$ (λ denotes the flow's rate. t_1, t_2 are link 1 and 2's propagation delays.) And the number of returning ACKs is also $\frac{t_1}{\frac{1}{\lambda}} + \frac{t_2}{\frac{1}{\lambda}}$. If link 2 is the bottleneck link, the other packets in the window will be queued or transmitted at r_1. If the flow does not change its window size, the state will persist. The same number of packets and ACKs will always be distributed along the pipe and the queue. Since we know link 2's capacity and we determine that it is the bottleneck, we conclude that the flow's steady state sending rate λ is equal to the link's capacity. Then we know immediately how many packets are in the pipe and how many packets are at r_1 without packet by packet level simulation.

In general, if we know which links are bottleneck links, we can always calculate the output queue length by a similar computation[2]. It gives us an easy way to obtain the steady state information.

Before explaining our approach for more general cases, we define some symbols.

W_j: window size of flow j. P_j: the static round trip delay of flow j, which is the sum of the link's propagation delay and transmission delay along the path of flow j. Because packets are of the same size, one link's transmission delay for a packet is a constant. We can view the transmission delay as an extension of the pipe's length. λ_j: the steady state rate of flow j. p_j: the path of flow j, which contains links flow j crosses. N_i: number of backlogged packets at outgoing link i. If the link i is a congested link, $N_i > 0$. Otherwise, $N_i = 0$. n_{ji}: number of backlogged packets from flow j at link i. If the link i is a congested link, $n_{ji} > 0$. Otherwise, $n_{ji} = 0$. C_i: capacity of link i.

According to the assumptions, we have:

For a flow j,

$$W_j = \frac{P_j}{\frac{1}{\lambda_j}} + \sum_{i \in p_j} n_{ji} \tag{1}$$

For a link i,

$$N_i = \sum_{j \in i} n_{ji} \tag{2}$$

Nevertheless, in the general case, we usually do not know each flow's steady state's sending rate. In fact, it is an unknown we want to compute. Thus, from Equation 1 and 2, we can not easily get n_{ji} nor N_i. However, λ_j and n_{ji} are related. They satisfy a set of physical constraints. The extra constraints provide a way to find out rates of flows and queue lengths at congested links.

As we know, if link i is not congested,

$$n_{ji} = 0, \ \forall j \in i \tag{3}$$

$$N_i = 0 \tag{4}$$

If link i is congested, the link must be sending at its full capacity. So,

$$C_i = \sum_{j \in i} \lambda_j \tag{5}$$

[2]In this paper, a bottleneck link refers to a link which is fully loaded. Sometimes we call it a congested link because such a link usually has a backlogged packet queue. However, it is possible in some case, a link is fully utilized but not congested. We do not distinguish them strictly.

If link i is not congested,

$$C_i \geq \sum_{j \in i} \lambda_j \tag{6}$$

In the steady state, if a link i is congested, each flow j across link i has a constant number of packets queued at it. The more backlogged packets one flow has at link i's output queue, the more chance it has to get a packet transmitted; hence the more throughput the flow could achieve. This observation follows because all packets are of the same length. More clearly, we have:

$$\frac{n_{ji}}{N_i} = \frac{\lambda_j}{C_i}, \ \forall N_i \neq 0 \tag{7}$$

Notice we can use Equation 7 to substitute n_{ji} in Equation 1, we finally get:

$$W_j = \lambda_j (\sum_{i \in p_j} \frac{N_i}{C_i} + P_j) \tag{8}$$

which is exactly the result from the Little's Theorem [1]. We illustrated it in an intuitive way.

Equation 5 and 8 are fully constrained. The total number of unknowns are the sum of the number of flows and the number of congested links. Corresponding to each flow, we have an Equation 8. And for each congested link, we have an Equation 5. The number of unknowns and the number of equations are matched.

Given a network configuration and each flow's window size, we believe the steady state is deterministic. Thus we claim in steady state, the rate of each flow and the queue length at each outgoing link must satisfy the above equation set and all the other constraints such as Inequality 6 and Equation 4. On the other hand, if a solution satisfies all the equal and unequal constraints, it must be the steady state parameters.

However, we still face a difficult problem. We do not know how to find out a solution for the model. The model consists of multidimensional non-linear equations 8 subject to linear constraints 5, 6, 4. If we can identify the congested links, therefore correctly choosing the right set of linear constraints, the problem will be reduced to finding a root for a set of non-linear equations. It is much easier. We will discuss the algorithm for identifying the congested links in the next section.

IV. Algorithm

A. The Difficult Point

The model consists mainly of two parts. The first part is:

$$W_j = \lambda_j (\sum_{i \in p_j} \frac{N_i}{C_i} + P_j) \tag{9}$$

In the second factor of the right hand side, the first term $\sum_{i \in p_j} \frac{N_i}{C_i}$ is the total queuing delay. The second term is the static round trip delay. The sum of the two is the average round trip delay of this flow. Thus this part of the model states the fact that in the steady state, a flow sends a window size of packets every round trip time. The flow's round trip delay is determined by its own path and the congestion condition of the network, which depends on the total load of the network.

The second part is:

$$\begin{cases} N_i &= 0 \\ C_i &\geq \sum_{j \in i} \lambda_j, \text{ if link } i \text{ is uncongested} \end{cases}$$

$$\begin{cases} N_i &> 0 \\ C_i &= \sum_{j \in i} \lambda_j, \text{ if link } i \text{ is congested} \end{cases}$$

The second part states the feasibility constraints. A link can not send faster than its capacity. If the incoming flows require more transmission rate than what the link can handle, their packets are backlogged. As flows are window controlled, they can not put more than a window size of packets into the network. Thus flows can not send faster than the returning rates of ACKs. As a result they will not further overflow the link. The number of backlogged packets are stable and the link will send at its full speed because every packet it transmits from the queue will trigger a new incoming packet.

As mentioned in Section III-A, window flow control does not guarantee a minimum sending rate nor a minimum delay. Without changing the window size, the flow's rate could become very low while the round trip delay becomes very huge. However, as the total number of packets inside the network is fixed, which is the sum of all flows' window sizes, the queuing delay is determined by both this number and the network configuration. Therefore, each flow's queuing delay can not be arbitrary. If we can figure out how packets are distributed along the pipe and backlogged at each output queue, we can calculate the queuing delay and further more, the steady state rates. We know that packets are backlogged if and only if the outgoing pipe is full, which is described by the second part of the model. Combining the two parts, we can find a set of steady state parameters. The difficult point is how to locate fully utilized links. The rates of flows are shaped by those links, which is the same phenomenon as TCP's self-clocking mechanism [5].

There is a simple and attractive idea. For each flow j, the maximum rate it could reach is $\frac{W_j}{P_j}$. This happens when there is no congested link along its path. This rate can be viewed as the maximum rate of this flow. For each link, if the sum of maximum rates from all incoming flows exceed its capacity, we could conclude the link is the congested link. However, this is not true. Since the real rates will be reduced by congestion, a flow might achieve a much lower sending rate than the maximum one. Thus, some link may seem to be congested but in fact it will not be. Even the seemingly "most congested link" may turn out to be uncongested at all.

B. How to Find a Solution for the Model

We developed an algorithm to find a valid solution for the model. When the network has reached a steady state, if we "remove" all the backlogged packets and reduce each flow's window size correspondingly, the network will still stay in the steady state and each flow will remain the same sending rate. This is the threshold for the network to go from uncongestion to congestion. At this point, though some links are sending at their full speed, they do not have backlogged packets. If we can catch this threshold, we can easily get each flow's steady state sending rate since there is no queuing delay yet and the round trip delay of each flow is equal to the static round trip delay. Starting from the threshold, if we inject packets into the network by increasing flows' window sizes to their original ones, packets will be queued up only at those links which are sending at the full speed at the threshold point. That is, the congested links in the steady state are the links that are sending at the full speed at the threshold point. This idea provides us some insight. We can add a light load to the network at the beginning. Next, we increase the load gradually and see whether there are some links that are reaching their capacities. We keep a record of those links until we increase the load to the given value. We use our record and the equation set of the model to get the steady state parameters. Here are the details of this algorithm.

We multiply each flow's window size by a scaling factor α ($\alpha \in [0,1]$) and substitute W_j in Equation 9 with αW_j. Let $Cong(\alpha)$ denotes the set of congested links for the value α. Obviously, $Cong(0) = \Phi$. Let $E(\alpha)$ be a set of equations chosen from the constraints. Let n be the number of flows. At each value of α, $E(\alpha)$ has n equations of the form:

$$\alpha W_j = \lambda_j \Big(\sum_{i \in p_j \text{ and } i \in Cong(\alpha)} \frac{N_i}{C_i} + P_j \Big) \tag{10}$$

Other equations in $E(\alpha)$ are of the form:

$$\sum_{j \in i} \lambda_j = C_i, \ \forall i \in Cong(\alpha) \tag{11}$$

The algorithm works as follows.

- Step 1: increase α to $\alpha + \delta$.
- Step 2: solve $E(\alpha + \delta)$ assuming $Cong(\alpha + \delta) = Cong(\alpha)$.
- Step 3: check the feasible constraints and update $Cong(\alpha)$ to $Cong(\alpha + \delta)$ based on the following rules. For link $i \ni$[3] $Cong(\alpha)$, if $(\sum_{j \in i} \lambda_j - C_j) \geq 0$, update $Cong$ by adding i into it. For link $i \in Cong$, if $N_i \leq 0$, delete it from $Cong$. If neither happens, *i.e.*, no link is overloaded and no queue size is below zero, $Cong$ remains the same.
- Step 4: update E based on the current $Cong$.
- Step 5: if $\alpha = 1$, solve E and stop. Otherwise, go to Step 1.

By induction, we prove that when $\alpha = 1$, $Cong(1)$ is the set of congested links when every flow j 's window size is W_j. Therefore, the feasible solution of $E(1)$ is the solution of the original equations and inequalities of the model.

Initially, because $Cong(0) = \Phi$, $E(0)$ only contains

$$0 \times W_j = \lambda_j P_j, \ j = 1, 2, \ldots, n \tag{12}$$

The solution of $E(0)$ is $\lambda_j = 0$ for any flow j. Because no link is congested, $Cong(0) = \Phi$ is the correct congested link set.

Suppose when α reaches some value, $Cong(\alpha)$ is the right set of congested links. We prove when α is increased by a tiny value δ, our algorithm will correctly update $Cong(\alpha)$ and as a result the solution of E will give the steady state parameters for the situation each flow j 's window size is reduced to the new value of αW_j.

When α is increased by a tiny value, inside the network, three possible situations could occur.

[3] $\ni$ means 'not a member of'.

First, the old congested links remain congested. The newly added loads are either queued at those old congested links or make other non-full pipes more crowded but not full, which means, no new congested link occurs. In this case, the set of congested links does not change. If we use the old $E(\alpha)$ set to get a new solution for $\alpha + \delta$, there will be no overflow for uncongested links and the old queue sizes are still positive since the old links are still congested. We do not change $Cong$ in this case. And $Cong$ is still the correct set of congested links for the new value of α.

Second, the new load makes some pipe full. Correspondingly, some new congested link occurs. As the old $Cong$ does not contain the new congested link, the solution of E based on the old $Cong$ will not satisfy $(\sum_{j\in i} \lambda_j - C_j) \leq 0$ for the new congested link. So, if we find out some constraint is violated, we can conclude the pipe must be filled up by the new α and we add the link to $Cong$.

Third, the new load makes some congested link uncongested. It sounds impossible at first. How could it happen while we add more load that a congested link becomes uncongested? The answer is that when a flow puts more packets into a congested link, all flows sharing the same link will suffer more queuing delay and as a result, their sending rates will be slowed down. Consequently, if those flows go through other links, they add less load to those links. Then it is possible for some formerly congested link to change to uncongested. When we solve E with the new value of α and the old set of congested links, we tend to get a very small queue length or a negative queue length. Because the link is in the congested link set, which means we assume its pipe is full, if we still get a positive queue length, the pipe can not accommodate all packets. So the link should still be congested at that point. The negative queue length implies the link should not be congested for the new α. We delete it from $Cong$.

As we update $Cong$ correctly in according to all possible situations, $Cong$ is still the right set of congested links.

Then we induce that $Cong(1)$ is the right set of congested links when $\alpha = 1$ and finish the proof.

C. Another Perspective of the Algorithm

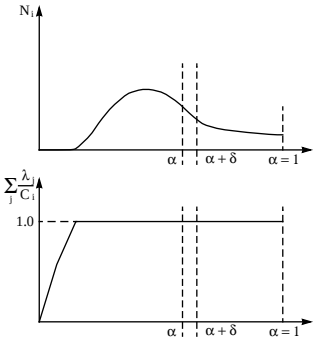

Fig. 2. N_i, $\frac{\sum_{j\in i} \lambda_j}{C_i} \sim \alpha$

The algorithm can be viewed from another perspective. Each flow's rate and each link's queue length are functions of α. They change dependently. As shown in Figure 2, 3 and 4, when the sum of flows' rates at a link is leveled out by the link's capacity, the link's queue starts increasing from zero. When the queue is drained out and becomes zero, the link is underloaded after that. The functions are non-smooth however. They are described by different formulas in different intervals of α. This is the reason why the equation set $E(\alpha)$ is also a function of α. But the changing points of a link's queue length and its total load are the same. At the changing points, both the left side's constraints and the right side's ones are valid. For example, if at a value of α_1, a link i starts to be congested, we can either insert i into $Cong(\alpha_1)$ or not. Because $N_i = 0$ and $\sum_{j\in i} \lambda_j = C_i$ are both true when $\alpha = \alpha_1$, if we insert i into $Cong(\alpha_1)$, and add unknown N_i and Equation 11 of C_i into $E(\alpha_1)$, the right solution has to satisfy $N_i = 0$; if $Cong(\alpha_1)$ has not contained link i yet, since we force $N_i = 0$, the right solution of $Cong(\alpha_1)$ must also satisfy $\sum_{j\in i} \lambda_j = C_i$, otherwise, $N_i = 0$ and $\sum_{j\in i} \lambda_j = C_i$ can not be both true, which is contradictory to the feasibility constraints. Similarly, if at some point, link i begins to be underloaded, whether we keep it in the $Cong(\alpha)$ or delete it will both satisfy the constraints.

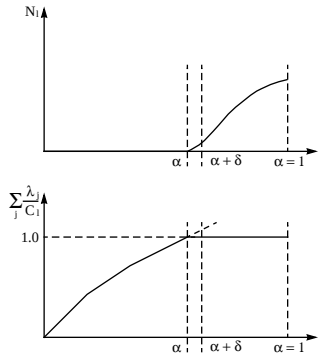

Fig. 3. N_l, $\frac{\sum_{j\in i} \lambda_j}{C_l} \sim \alpha$

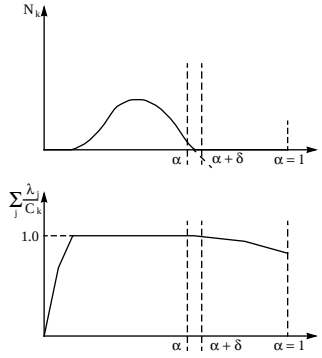

Fig. 4. N_k, $\frac{\sum_{j\in i} \lambda_j}{C_k} \sim \alpha$

Though the rates of flows and the queue lengths of links are non-smooth functions of α, for each value of α, $E(\alpha)$ consists of smooth continuous functions of α. By the continuity of a smooth function, if for some value of α, $\sum_{j\in i} \lambda_j = C_i$ is true, within a small neighborhood of α, such as $[\alpha-\delta, \alpha+\delta]$, $C_i - \epsilon < \sum_{j\in i} \lambda_j < C_i + \epsilon$ is also true. Figure 3 gives an example. We assume at point α, link l is not congested. The induction assumption says it is correct. For this value of α, the set of

constraints in E is a set of continuous smooth functions. If at the two end points of the interval $[\alpha, \alpha+\delta]$, $\sum_{j\in i}\lambda_j$ changes from less than C_l to over C_l, we deduce there must be a $\alpha' \in [\alpha, \alpha+\delta]$, which satisfies $\sum_{j\in l}\lambda_j = C_l$. As discussed above, this indicates the changing point of the congestion condition of link l. By adding link l into $Cong(\alpha+\delta)$, we correctly identify the new congested link. The same analysis is applicable when a link changes from congested to uncongested. As the solution of E guarantees the load of a congested link is its capacity, the change of queue length is used as the indication instead. When a congested link's queue length goes to negative, it implies the link is going to be uncongested under the new value of α (See in Figure 4). In both cases, the solutions of $E(\alpha)$ assuming $Cong(\alpha) = Cong(\alpha+\delta)$ are the extrapolation of the set of constraint functions in $E(\alpha)$ (Figure 3 and 4).

V. Short Discussion About the Implementation

A. Two Major Concerns

There are two major concerns about the implementation. The first concern is the number of iterations. Our algorithm relies on the continuousness of constraint functions in E. A large increment of α will cause discontinuous change of congested links and result in incorrect updates of the congested link set. Too small increment will increase the number of iterations and reduce the efficiency of the model. Thus we employ the idea of a binary root finding algorithm to solve the dilemma. When α is far away from some changing point of the congested link, we increase α in large steps. When α is close to some changing point of the congested link, we reduce the increment of α by half each time until we could correctly update the congested link set under the increment of α.

Another technique to reduce the number of iterations is to choose a better starting α than $\alpha = 0$. At the starting point, we assume all links are uncongested. The equation set E consists of linear independent equations. We can get $\lambda_j = \alpha W_j/P_j$ directly. Using the capacity constraint, $\sum_{j\in i}\lambda_j \le C_i$. we have

$$\alpha \le \frac{C_i}{\sum_{j\in i}(W_j/P_j)} \tag{13}$$

The maximum α that satisfies this condition will be a good starting point for α. We do not need to increase α from zero.

Another concern is how to solve the set of equations in $E(\alpha)$. As a matter of fact, there are no good, general methods for solving systems of more than one nonlinear equations [6]. Finding a root for a set of nonlinear equations which have N unknowns is to find points which are common to N unrelated zero-contour hyper-surfaces(the N equations). Each equation is of dimension $N-1$ in a N dimension space. The numerical methods for solving such equations usually involve iterations and matrix operations. Convergence is always a problem. The Newton-Raphson Method is the simplest multidimensional root finding method. We used the code from the book *Numerical Method* [6] for solving the equations. Given the correct set of congested links, the Newton-Raphson method converges very quickly to a solution for the set of equations in E.

B. Performance Analysis

The major cost of the model is solving the equations. If E contains k valid equations, the time needed to solve the equations is $O(\frac{1}{3}k^3)$. As the number of congested links can not exceed the number of flows, k is at most $2n$ (Remember n denotes the number of flows.). It is good that this number is not related to the size of the network. Only the number of flows we want to study matters. Unlike simulation, the cost of solving the equations does not increase with the number of links a flow crosses.

The number of iterations used to identify the congested links depends on the network's configuration and the total load. In general cases, a link's congestion state will stay stable within a large range of load, which means, with the increase of α, a link does not switch from congestion to non-congestion and vice versa very rapidly. Because the algorithm tries to detect all switching points, if there are not many of them, the number of iterations is moderate.

If we want to further speed up the model, we can optimize the solving of the equations by using a package. Because our focus in this paper is on the idea rather than the technique, we have not evaluated other equation solving methods. Our goal is to show the validity of the model. We developed the algorithm and the implementation in order to test the model.

VI. Results Comparison with those of *ns 1.0*

We ran the same test nets on *ns 1.0* and compared the results achieved by modeling and simulation. In all cases, the model gives very similar results. It is hard to compare the efficiency of the model and the simulation by *ns* because *ns* is implemented via Tcl interface, which makes it slower.

Here we give one typical test result. The testing net is based on a network topology generated by the software *gtitm* [7]. The net is hand edited for our need.

The net has 109 links and 110 nodes. We have 21 flows running simultaneously. Each flow has a window size of 20. The test net contains 5 domains. Each domain contains 3 local nets. Links connecting the end user to the subnet are of capacity 8.0Mbps. Links connecting intra-localnet routers are of capacity 1.6Mbps. Links connecting inter-localnet routers are of capacity 2.4Mbps. And links connecting inter-domain routers are of capacity 3.2Mbps.

The results comparing for queue length is shown in Figure 5. Only the congested links are listed. The results comparing rates are shown in Figure 6.

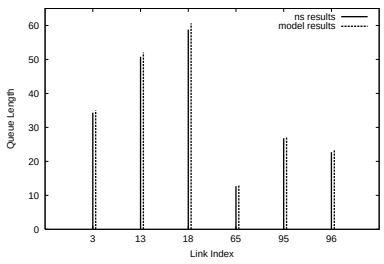

Fig. 5. Results Comparison: Queue Lengths

On a Pentium 166MHz computer with Free-BSD OS, it roughly takes the model 0.11sec to compute the results. For

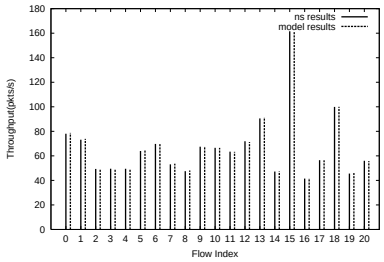

Fig. 6. Results Comparison: Average rates of flows

ns, we design the simulation time to be 100 seconds. It takes *ns* about 65 seconds.

VII. Application Results

This section discusses the model's application. Since in real routers, buffer sizes are limited, when a router's buffer overflows, packets get dropped. TCP will adjust its window size correspondingly. In this section, we assume links are using RED [2] dropping mechanism. We simulate TCP with RED links by applying this model. In Section VII-A, we discuss the assumptions our simulation depends on. In the next two sections, we discuss the simulation methods and results.

A. Assumptions

In the assumptions in Section III, we required each flow's window size to be fixed. However, in a net with RED links, a TCP flow adjusts its window size according to the congestion condition inside the net. Thus the load of the network changes dynamically. The assumption about steady state loses effect. We exploit the ideas of event driven simulation to simulate the dynamic changes. Still, we ignore the detailed implementation of TCP and RED. Our simple protocol does not consider the slow start, multiple drops and time-out. The parameters we are interested in is the long term average rate a flow could get. We assume a flow's window size is determined by its congestion window. A flow opens its window size by one per round trip time if it does not suffer a packet drop. If a flow suffers at least one packet drop, it cuts its window by half. The model is used as a "submodel" to compute the average queue lengths and each flow's rate according to the instanteous window size of each flow.

B. The Event Driven Simulation

B.1 The Simulation Method

In our simulation, an event refers to a change in a flow's behavior. Either a flow increasing its window size or decreasing its window size is counted as an event. When an event happens, we calculate the current queue length at each router assuming the net has achieved steady state. We use the queue length computed from the model as an approximation of the average queue length a RED router computes. We run the RED algorithm based on the current queue length. Since in each round trip time, a flow has a window size of packets distributed across every link along its path, a window size of packets of this flow will pass the RED filter at each congested link. If any of them get dropped, we schedule the next event after the flow's current round trip time with its window cut by half. If none of them get dropped, we schedule the next event after the flow's current round trip time with its window increased by one. The parameters for RED are $max_{th} = 25$, $min_{th} = 10$ and $max_p = 0.2$. [2]

For the queue length computing procedure, we can use the algorithm in Section IV to solve the equations directly. That is, we assume all links are not congested at first and we iterate through α until we find the final solution. However, if there is no packet drop during last time interval, during this time interval, only one flow's window size is increased by one. In this case, links' congestion states should not vary a lot. Links that were congested during the previous time interval are probably still congested and the uncongested links are still uncongested. We then try last time's congested link set directly. If this gives us a solution feasible and satisfying all the other constrains, we are set. If this does not, we invoke the original procedure as a back up.

We simulated the situation that a flow detects a packet drop within a single round trip time. Multiple drops in one round trip time are dealt with in the same way as a single drop. We assume this situation is applicable to the real networks.

B.2 Results

We tested the model's simulation results with those of *ns 1.0*. We designed two test nets. The first one is the test net introduced in Section V. The other one is a high speed net with the same topology but each link's capacity is increased by a factor of 10. We also have 21 flows. The results are listed in Figure 7 and Figure 8.

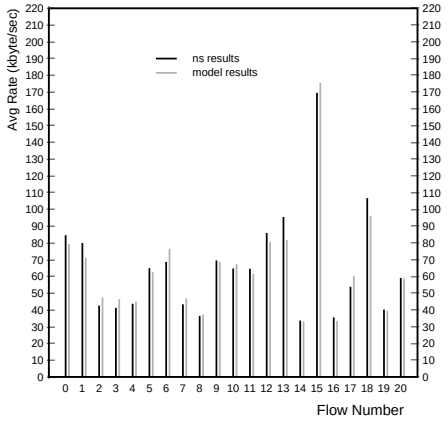

Fig. 7. Simulation Results of a Low Speed Net

Compared to *ns* , the average error of the results for the low speed net is 6.4%. The one for the high speed net is 9.0%.

We are very interested in the speed of the two methods. In the low speed net test, *ns* wins. It takes *ns* about 65 seconds to do a 100 seconds simulation. For the model, it costs about 110 seconds. In a low speed net, each flow can only send a small window size of packets every round trip time. Every state of a packet causes an event. Since the cost of packet by packet event driven simulation is proportional to the number of events, if the total number of events is rather small, *ns* is not slow.

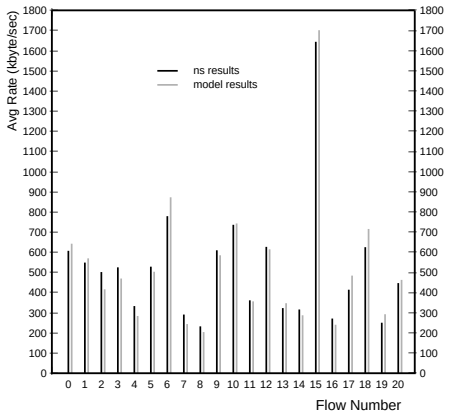

Fig. 8. Simulation Results of a High Speed Net

However, in the test of a high speed net, it takes *ns* about 1.3 hours to do a 100 seconds simulation. For the model, the cost is about 11 seconds. The increase of the cost of *ns* is due to the increase of each flow's window size, which results in the increase of events. A packet by packet event driven simulator has the limit of simulating high speed networks. The decrease of the cost of the model is due to two reasons. One reason is that in the high speed net with the same number of flows, the net is lightly loaded. The number of packet drops is reduced. A flow has more chances to increase its window size by one than it has in a low speed net. Also, it has less chance to cut its window size by half than it has in a low speed net. In this case, links' congestion states do not change dramatically. When an event happens, if we assume the set of congested links is the same as the last event, the assumption is right in most cases. So we do not need to invoke the original α iteration algorithm. The other reason is if a flow with a large window cuts its window size by half, it will greatly reduce the congestion of the network. The network may change from a slightly congested to a uncongested state. For a uncongested or a net with a few congested links, the cost of solving the equations is reduced. As the model's cost is mainly increased with the number of flows, unlike *ns*, the increase of window sizes does not increase its cost. Therefore, the modeling result of a high speed net is much faster than the low one.

We also tried discrete time driven simulation. It shows that the results of time driven simulation is very sensitive to the time interval we chose. When the RTTs of flows fall into a narrow range, the results of the model is also very close to those of *ns* .

VIII. Summary and Future Work

A. Summary

This paper introduces a model for studying the steady state of window based flow control packet-switched networks. We construct the model from the observation of TCP's steady state behavior. The model includes the feature of window based flow control mechanism and feasible link capacity constraints. The test results show that the model is a correct abstraction of the real window flow control packet-switched network. We further apply this model to simulate window flow control networks with RED routers. We compared the results from the model and those from *ns* . We do not mean the model can replace the simulation. Simulation is a more accurate way to obtain dynamic state information of networks. However, if we want to get a quick estimation of the long term average statistics, the model's results should be sufficient.

B. Future Work

There are several remaining questions. First of all, the algorithm for finding a solution of the model needs further improvement. We are interested in collecting more information from the network in order to simplify the model. The only hard problem is how to identify the congested links when each flow's window size is given. We are not sure whether we can achieve that without doing iterations of solving non-linear equations.

The second question includes developing more applications of the model. We chose RED because RED routers drop packets based on the average queue length, which is close to the queue lengths the model computes. We want to find out whether the model can be used to simulate other dropping mechanism.

One defect of the model is that it does not incorporate the dynamics of TCP explicitly[4]. It has to depend on event driven simulation when modeling TCP's window's adaptation algorithm. To totally avoid event driven simulation, we could use the "TCP-friendly" equation [8] [9] to develop a different set of equations.

For each link, the feasibility constrain (Equation 6) still holds.

For each flow, the "TCP-friendly" equation shows:

$$\lambda_j = \frac{1.3}{RTT_j\sqrt{\rho_j}} \tag{14}$$

Another equation showing the relation of RTT and loss rate ρ is also needed. We are currently working on this part.

IX. Acknowledgment

The problem was brought up by my advisor David Clark. Without his guidance, this work would not be possible. I also thank Sally Floyd for her useful feedback on my work and thank Dorothy Curtis for proofreading the final version of this paper.

References

[1] Dimitri Bertsekas and Robert Gallager, *Data Networks*, Prentice-Hall, Inc, 1992.
[2] Floyd S. and Jacobson V., "Random Early Detection gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, Aug. 1993.
[3] http://www nrg.ee.lbl.gov/ns/, "ns version 1 - LBNL Network Simulator," .
[4] Anlu Yan and Wei-Bo Gong, "Fluid Simulation for High Speed Networks," in *Proceedings of the 15th International Teletraffic Congress*, June 1997.
[5] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of the ACM SIGCOMM'88*, Aug. 1988.
[6] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in C*, the Press Syndicate of the University of Cambridge, 1992.
[7] Ken Calvert and Ellen Zegura, "GTITM," .
[8] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic.," *Computer Communication Review*, vol. 21, no. 5, Oct. 1991.
[9] Jamshid Mahdavi and Sally Floyd, "TCP-Friendly Unicast Rate-Based Flow Control.," *http://www.psc.edu/networking/papers/tcpfriendly.html*, Jan. 1997.

[4] This comment is from Sally Floyd.