FAST AND STRONG CONTROL OF CONGESTION-MAKING TRAFFIC

Gaeil Ahn, Kiyoung Kim, Jongsoo Jang

Security Gateway Research Team, Electronics and Telecommunications Research Institute (ETRI)
161 Gajeong-dong, Yuseong-gu, Daejon, 305-350, Korea

Keywords: Network congestion, DDoS attack, Packet marking, Priority queue, Traffic control

Abstract: In case that malicious or selfish user congests network, the traditional congestion control schemes such as ECN (Explicit Congestion Notification) in TCP protocol could not control the pernicious congestion so perfectly as they protect normal traffic. In this paper, we propose a strong congestion-making traffic control scheme, which is capable of preventing malicious or selfish user from congesting networks by dropping only packets corresponding to congestion-making traffic when a network congestion occurs. Our scheme involves two mechanisms: a traffic service decision mechanism that is able to fast and correctly determine whether an incoming packet is normal traffic or congestion-making, and a marking mechanism for identifying congestion-making traffic. In the marking mechanism a router can mark a packet in order to notify downstream routers that the marked packet is congestion-making traffic. To show our scheme's excellence, its performance is measured and compared with that of the existing schemes through simulation.

1 INTRODUCTION

Congestion at a router occurs when the sum of input streams is greater than output capacity. Once congestion occurs, packet delay and packet loss increase because congested router's buffers become full. In solving the congestion problem, most of the traditional mechanisms assume that sender makes an effort to adjust its transmission rate to network state or reserves a certain amount of network resource before sending packets. For example, network in ECN (Explicit Congestion Notification) mechanism (S. Floyd, 1994) notifies TCP senders/receivers about incipient congestion in expectation that if they receive an ECN signal they'll decrease their transmission rate.
However, malicious or selfish users may intentionally congest network. Actually, Denial of Service (DoS) attackers (K. J. Houle et al, 2001), (X. Geng et al, 2000) as malicious user result in network congestion by generating a huge volume of traffic for the purpose of assailing a victim system or networks. Selfish users also establish many connections to get better network service. In that case, ECN-like approach would be of little use in controlling such pernicious congestion. That is, when congested router sends an ECN signal to senders, only normal senders will decrease its transmission rate. On the other hand, malicious or selfish users will ignore the ECN signal by increasing the number of flows. This is why network is still congested nevertheless normal user's decreased its transmission rate.

In this paper, we define congestion-making traffic as one of which transmission rate is still high under network congestion. To control congestion-making traffic, it has been proposed a static rate-limit scheme (Cisco, 2000) in which the rate to limit is fixed in advance. But the scheme has a disadvantage that the amount of packets during normal state should be first measured to set the correct threshold value for rate-limiting congestion-making traffic. And also, it has been proposed a dynamic rate-limit scheme called ACC (Aggregate-based Congestion Control) (R. Mahajan et al, 2002) in which when congestion occurs the rate to limit is dynamically determined based on the bandwidth of each aggregate. Even if ACC is much better than static rate-limit, it is likely to have a weak point in determining precise rate-limit value. As explained above, TCP-like adaptive traffic decreases its transmission rate when congestion occurs. So, the threshold for rate-limiting congestion-making traffic may be overestimated. As a result, ACC may not protect adaptive traffic from congestion-making traffic.

In this paper, we propose a strong congestion-making traffic control scheme for preventing malicious or selfish user from congesting networks. Our elementary strategy is to drop only packets corresponding to congestion-making traffic when network congestion occurs. We don't use rate-limit approach in controlling congestion-making traffic. Instead, we employ three priority queues: high, medium, and low priority queues. We provide normal traffic with high priority queue and congestion-making traffic with medium or low priority queue. Our scheme prevents the buffer of high priority queue from becoming full. So when network congestion occurs, only medium and low priority queues experience that their buffer become full. In other words, network congestion is localized to only congestion-making traffic without having any effect on normal traffic.

Our scheme involves two mechanisms: a traffic service decision mechanism that is able to fast and correctly determine whether an incoming packet is normal traffic or congestion-making, and a marking mechanism for identifying congestion-making traffic. In the marking mechanism, a router can mark a packet in order to notify downstream routers that the marked packet is congestion-making traffic. If a router receives a marked packet, it forwards the packet to low priority queue at once without determining the service queue for it.

The rest of this paper is organized as follows. Section 2 overviews our scheme and section 3 illustrates an algorithm for determining the service of an incoming packet in detail. In section 4, the performance of the proposed scheme is measured and compared with that of the existing schemes through simulation. In section 5, the other works that tackle the problems of DoS attacks is shortly described. Finally conclusion is given in Section 6.

2 CTC OVERVIEW

In this section, we describe our scheme, Congestion-making Traffic Control (CTC)

2.1 IP Spoofing Problem

Generally, severe network congestion occurs by malicious or selfish user. Malicious user is likely to employ the faked source IP to hide his/her identity, while selfish user does not use the faked source IP because his/her purpose is not to attack networks/systems, but to get better network service.

In this paper, we don't address IP spoofing problem. We assume that there is no IP-spoofed packet on networks or if it exists it is detected/dropped by using the existing mechanism such as ingress filtering (P. Ferguson et al, 2000), unicast reverse path forwarding (uRPF) (Cisco, 2001) and so on.

2.2 Identifying Congestion-making Traffic

Malicious or selfish user has a common characteristic that generates a huge volume of traffic without any consideration of network state. That is, they generate heavy traffic and never decrease their transmission rate even if network congestion occurs. For such reason, we define congestion-making traffic as one that generates heavy traffic under network congestion.

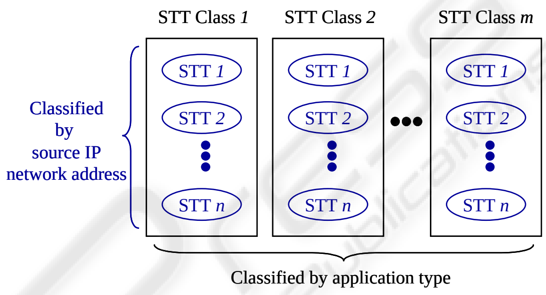

STT Class 1
STT Class 2
STT Class m
Classified by source IP network address
STT 1
STT 2
STT n
Classified by application type

Figure 1: STT and STT class

In this paper, we classify all traffic by its source IP network address and application type (i.e, destination port and protocol). More specifically, we define two terms, a source-based traffic trunk (STT) and a STT class as shown in Fig. 1. The original concept of traffic trunk was introduced by T. Li and Y. Rekhter (T. Li et al, 1998). In this paper, a STT indicates an aggregate of flows with the same source IP network address. The granularity of STT depends on the size of address prefix. A STT class indicates the set of STTs with the same application type. STT class can be defined with a policy. For example, it is possible to define STT class 1 as one for Web traffic and STT class 2 as one for voice traffic.

In this paper, we define normal traffic as one that consists of STTs with relatively less load than other STT, and of which the sum of load does not exceed *MaxLoad_Threshold*. *MaxLoad_threshold* can be configured by an administrator. The definition of normal traffic and congestion-making traffic is as follows :

$$\text{Normal Traffic} = \left\{ \begin{array}{l} STT_0 \,..\, STT_m \mid \left(\sum_{i=0}^{MAX\ m} Load\ of\ STT_i \right) < (MaxLoad\_Threshold), \\ \qquad Load\ of\ STT_i \le Load\ of\ STT_{i+1} \end{array} \right\}$$

$$\text{Congestion-MakingTraffic} = \{All\ Active\ STTs\} - \{Normal\ STTs\}$$

To fast and correctly distinguish between normal and congestion-making traffic, we propose a novel algorithm. The algorithm is described in section 3.2 in detail.

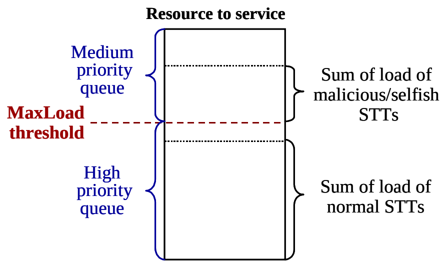


Figure 2: Our strategy for controlling congestion-making traffic

Our strategy for controlling congestion-making traffic is to keep the sum of load of normal STTs from exceeding *MaxLoad_threshold* and to provide them with better service than congestion-making STTs for the purpose of localizing the effect of network congestion to congestion-making STTs.

To realize this, we employ two kinds of priority queues, high and medium priority queues as shown in Fig. 2. We provide normal STT with high priority queue and congestion-making STT with medium priority queue. If the sum of load of normal STTs is greater than *MaxLoad_threshold*, we change the property of the worst one of the normal STTs to congestion-making traffic for the purpose of avoiding that network congestion occurs in high priority queue. On the contrary, if the sum of load of all normal STTs is less than *MaxLoad_threshold*, we changes the property of the best one of the congestion-making STTs to normal traffic for the purpose of increasing network utilization.

2.3 Architecture of CTC

We now describe our scheme. Fig. 3 shows the architecture of CTC-enabled router. The architecture is composed of two parts: packet forwarding and STT examiner.

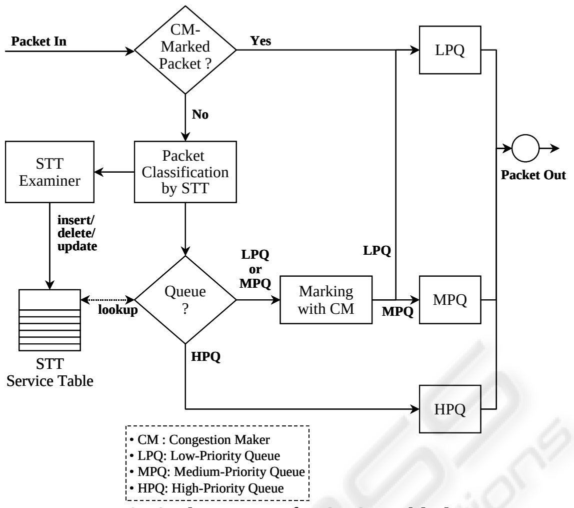


Figure 3: Architecture of a CTC-enabled router

Firstly, we explain about packet forwarding. When CTC-enabled router receives a packet, it checks if the packet is marked with congestion-maker (CM). If it is, CTC-enabled router sends it to Low Priority Queue (LPQ). Otherwise, it classifies the incoming packet by STT (i.e. source IP network address, destination port, and protocol) and looks up the service queue of the STT corresponding to the packet from STT Service Table. And then it sends the packet to High Priority Queue (HPQ), Medium Priority Queue (MPQ) or LPQ according to the lookup result. If the service queue of the packet is MPQ or LPQ, then the packet is marked with CM.

Secondly and finally, STT-Examiner takes the responsibility of determining whether a STT corresponding to an incoming packet is congestion-making traffic or not. STT-Examiner calculates the average bandwidth of a STT corresponding to an incoming packet and determines the service queue of the STT based on bandwidth of the STT and the sum of bandwidth of normal STTs. The operation result is stored in STT Service Table. Algorithm of STT-Examiner is described in section 3 in detail.

STT Service Table consists of several fields such as STT-Identifier, service queue by STT-Examiner, service queue by Intrusion Detection Agent (IDA), the average bandwidth of STT, and so on. Where, IDA means IDS (Intrusion Detection System)-like agent.

STT (/24 prefixes)	Service Queue by STT-Examiner	Service Queue by IDA
192.168.10.x	MPQ	•
192.168.21.x	HPQ	•
192.168.32.x	HPQ	LPQ
192.168.41.x	HPQ	MPQ
192.168.51.x	MPQ	•

Figure 4: Update of STT service table: can be done by IDA as well as by STT-examiner

Our architecture can support IDA in order to enhance the correctness when distinguishing between normal and congestion-making traffic. That is, STT Service Table can be updated by IDA as well as by STT-examiner as shown in Fig. 4. In that case, IDA has higher priority than STT-examiner in packet forwarding. For example, if IDA regards a STT (e.g., 192.168.32.x in Fig. 4) as malicious traffic and set its service queue to LPQ, then packets with the STT will be forwarded to LPQ irrespective of the service decision of STT-examiner.

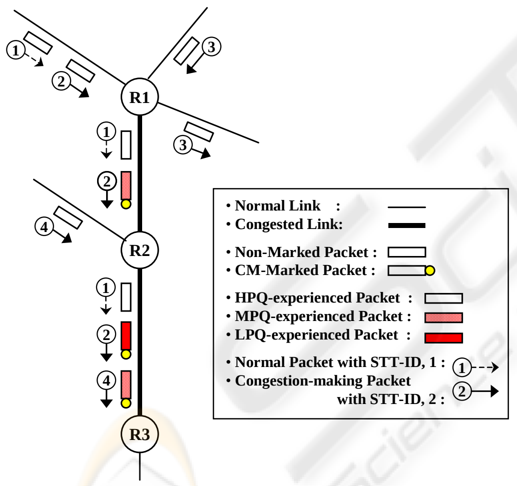

R1
R2
R3
①
②
③
④
• Normal Link :
• Congested Link:
• Non-Marked Packet :
• CM-Marked Packet :
• HPQ-experienced Packet :
• MPQ-experienced Packet :
• LPQ-experienced Packet :
• Normal Packet with STT-ID, 1 :
• Congestion-making Packet with STT-ID, 2 :

Figure 5: Example about how congestion-making STT is controlled

Fig. 5 shows how our CTC scheme controls congestion-making traffic. In the Fig. 5, the two links between R1 and R3 are congested. ①,②,③, and ④ indicate STT identifier. The normal packet with STT-ID, ① gets good Quality of Service (QoS) at both R1 and R2. The congestion-making packet with STT-ID, ②, however, gets bad QoS at R1 and worse QoS at R2 than at R1. So, The drop probability of the packet is very strong because it is forwarded using MPQ at R1 and LPQ at R2, respectively.

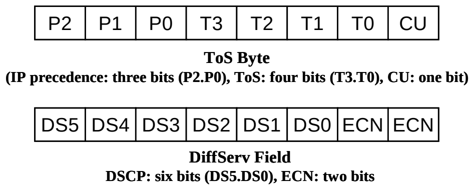

P2
P1
P0
T3
T2
T1
T0
CU
ToS Byte
(IP precedence: three bits (P2.P0), ToS: four bits (T3.T0), CU: one bit)
DS5
DS4
DS3
DS2
DS1
DS0
ECN
ECN
DiffServ Field
DSCP: six bits (DS5.DS0), ECN: two bits

Figure 6: Use of ToS field in IP Header by Diffserv

To support CM-marking proposed in this paper, we need a field in IP header. There is ToS field in IP header to show precedence and type of service for a packet. But, the ToS field is now used by DiffServ (Differentiated Service) (K. Nichols et al) as shown in Fig. 6. The six most signification bits of the ToS byte are now called the DiffServ field. The last two Currently Unused (CU) bits are now used as ECN bits. Until now, *DS0* in DiServ Field is always 0 (F. Baker et al) (V. Jacobson et al). So we have under investigation whether we can use *DS0* in doing CM-marking.

3 STT SERVICE DETERMINATION ALGORITHM

We now describe an algorithm for determining on the service for a STT in detail, which is executed by STT examiner explained in previous section.

3.1 Fluctuation-sensitive STT metering

To correctly determine which one is congestion-making traffic, it requires precisely calculating the average bandwidth of STT. However, it's not easy to get exact bandwidth of STT because packets in current TCP/IP Internet are forwarded from a router to a router at variable bit rate. To make it worse, malicious user may generate artificial congestion-making traffic with violent fluctuations in on-off sending pattern.

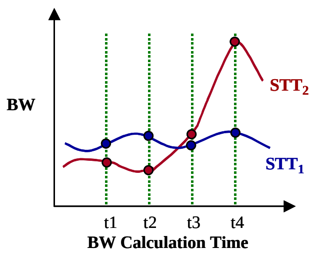


Figure 7: Congestion-making traffic (STT_2) with violent fluctuations

For Example, let's suppose that STT_1 and STT_2 as shown in Fig. 7 are normal traffic and congestion-making traffic, respectively. In Fig. 7, t1-t4 each indicates the time to calculate the average bandwidth of STT and to determine which one is congestion-making traffic. STT_2 is generating heavy traffic in an instant from t2 to t4 to bring out network congestion. In that case, if the bandwidth of both STTs is ordinarily metered, it'll be difficult to be aware at t3 that STT_2 is congestion-making traffic. This is because the average bandwidth of STT_2 is underestimated.

For fast detection of congestion-making traffic, we propose a fluctuation-sensitive metering scheme, which considers traffic increasing rate as well as current traffic volume in calculating the average bandwidth of STT. The proposed metering scheme makes use of Exponentially Weighed Moving Average (EWMA), which employs a statistic to average the data in a way that give more weight to recent observations and less weight to older observations. In this paper, the average bandwidth of a STT is calculated as follows :

$$STT.avgBW = STT.avgBW \times (1.0 - \alpha) + \\ (STT.sampleBW + STT.additionalBW) \times \alpha$$

Where, *STT.sampleBW* indicates the current bandwidth of STT calculated based on the total size of packets received during a certain given time. 0 < *alpa* < 1 and *alpa* is used to mitigate a sudden change of *STT.avgBW*.

The value of *STT.additionalBW* dedepnds on traffic increasing rate of the STT. If the traffic increasing rate (i.e. *STT.sampeBW* - *STT.avgBW*) is less than or equal to 0, then *STT.additionalBW* is 0. Otherwise, *STT.additionalBW* is calculated as follows:

$$STT.additionalBW = \frac{(STT.sampleBW - STT.avgBW) \times STT.sampleBW}{STT.avgBW}$$

3.2 Fast STT Service Determination

In this paper, we propose an algorithm as shown in Fig. 8, which is capable of determining fast if the service queue of a STT is HPQ or MPQ according to its average bandwidth.

The proposed procedure has four parameters: *STT*, *AvailableBW*, *STT_ToPromote*, *STT_ToDemote*. *STT* contains information about a STT that an incoming packet belongs to. *AvailableBW* indicates the available bandwidth for a STT class. In initial time, *AvailableBW* is set to the max bandwidth that can be allocated for the STT class. *STT_ToPromote* is one to promote foremost of all the MPQ-served STTs in case that one of them have to be changed to HPQ in service queue. And *STT_ToDemote* is one to demote foremost of all the HPQ-served STTs in case that one of them have to be changed to MPQ in service queue.

```
Procedure Fast-STT-SQ-Decision ( STT, AvailableBW,
                                  STT_ToPromote, STT_ToDemote )

// STT : Source-based Traffic Trunk that a incoming packet belongs to.
// AvailableBW : the available bandwidth for a STT class.
//              initially, it's set to the max bandwidth for the STT class
// STT_ToPromote : is one to promote foremost of all the MPQ-served STTs
//              in case that one of them have to be changed to HPQ in service queue.
// STT_ToDemote : is one to demote foremost of all the HPQ-served STTs
//              in case that one of them have to be changed to MPQ in service queue.

  if ( time to calculate the average bandwidth of STT ) then
     previousAvgBw ← STT.avgBw

     // calculate the average bandwdith of STT
     calculate-STT-BW (STT)

     if ( STT.sq = MPQ && STT.avgBw ≤ AvailableBW ) then
        STT.sq        ← HPQ
        AvailableBW ← AvailableBW − STT(i).avgBw

     else if ( STT.sq = HPQ ) then
        AvailableBW ← STT.avgBw − previousAvgBw
        if ( AvailableBW < 0 )
           STT.sq        ← MPQ
           AvailableBW ← AvailableBW − STT.avgBw
        end
     end
  end

  if ( STT.sq = HPQ ) then
     STT_ToDemote.avgBw ← max (STT_ToDemote.avgBw , STT.avgBw )
  else
     STT_ToPromote.avgBw ← min (STT_ToPromoote.avgBw, STT.avgBw )
  end

  if (STT_ToDemote.avgBw > STT_ToPromote.avgBw ) {
     AvailableBW ← AvailableBW +
                    (STT_ToDemote.avgBw − STT_ToPromote.avgBw)
     STT_ToDemote.sq ← MPQ
     STT_ToPromote.sq ← HPQ
  end

END Fast-STT-SQ-Decision
```

Figure 8: Algorithm for determining on the service queue of a STT

The proposed algorithm, *Fast-STT-SQ-Decision* is executed whenever a packet arrives. Firstly, the algorithm checks if it's time to calculate the average bandwidth of the STT corresponding to the packet. If yes, it calculates the average bandwidth of the STT. If the service queue of the STT is MPQ and there is available bandwidth for it, then its service queue is changed to HPQ. On the contrary, Even if the current service queue of the STT is HPQ, if the increased bandwidth of the STT is greater than available bandwidth then its service queue is changed to MPQ. Without regard to STT bandwidth calculation time, the proposed algorithm is always trying to set *STT_ToPromote* to a STT consuming the least bandwidth of MPQ-served STTs, and to set *STT_ToDemote* to a STT consuming the most bandwidth of HPQ-served STTs. If the average bandwidth of *STT_ToDemote* is greater than that of *STT_ ToPromote*, the service queue of *STT_ToDemote* is changed to MPQ and that of *STT_ ToPromote* is changed to HPQ.

The proposed algorithm is capable of satisfying the definition of normal and congestion-making traffic defined in section 2.2 within a fast time without using time-consuming operation such as sort. We'll show how fast and correctly the proposed algorithm can determine whether a STT is normal STT or congestion-making through a simulation in next section.

4 SIMULATIONS

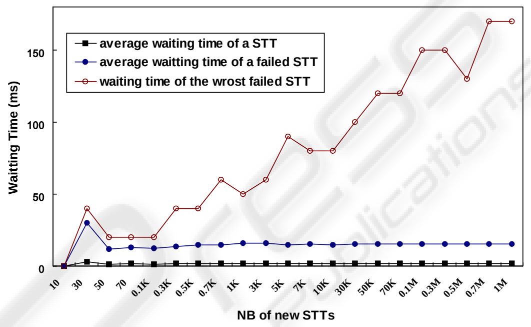
4.1 STT service queue determination algorithm

The purpose of this simulation is to see how fast the proposed algorithm (i.e., Fast-STT-SQ-Decision algorithm of Fig. 8) can determine the service queue for a STT. For the simulation, we assume that the bandwidth of each STT is fixed at simulation initial time and the packet inter-arrival time of each STT is 10ms. The simulation finishes when the proposed algorithm provides all normal STT with HPQ and all congestion-making STT with MPQ.

Fig. 9 shows simulation results of the STT service queue determination algorithm proposed in this paper. In Fig. 9, <average waiting time of a STT> means the average time taken to correctly determine the service queue of a STT. In <average waiting time of a failed STT>, a failed STT means one that fails to find out its service queue at one time. And in <waiting time of the worst failed STT>, the worst failed STT indicates one that most fails to find out its service queue. In the simulation of Fig. 9-(a), we fix the percentage of congestion-making STT to 60%. In the simulation of Fig. 9-(b), we fix the number of STTs to 10000.

The performance of our algorithm is very excellent as shown in Fig. 9. Our algorithm is little influenced by both the total number of STTs and the percentage of congestion-making STTs.

(a) Increase in the total number of STTs



(b) Increase in the percentage of congestion-making STTs

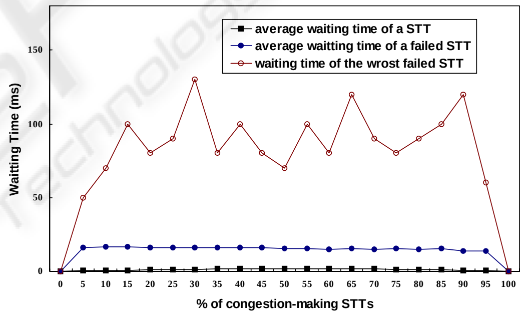


Figure 9: Simulation results of the STT service queue determination algorithm

Fig. 9 means that once the average bandwidth of a STT is calculated, our scheme can correctly determine on the service queue of the STT within 2ms (the average time). In Fig. 9, the average waiting time of a failed-STT is about 25ms. The waiting time, 25 ms is not long in this simulation because it means that only two packets (since the inter-arrival time is 10ms) are forwarded to wrong queue.

4.2 CTC

In order to compare performance between our scheme and the existing algorithms, we use ns-2 Network Simulator (UCB/LBNL/VINT). In this simulation, the priority queues of our scheme are implemented using CBQ (Class-based Queuing).

Fig. 10 shows the network topology for simulations. The topology consists of six source nodes from 0 to 5, three router nodes from 6 to 8, and two target nodes from 9 to 10. In the source nodes, the node 0-3 and 5 mean normal user and the node 4 means malicious/selfish user. In this simulation, we consider three kinds of traffic type: adaptive TCP, non-adaptive UDP, and Web-like traffic.

Firstly, the simulation scenario for adaptive TCP and non-adaptive UDP traffic is as follow. The node 0 and 2 each has three non-adaptive UDP flows of which each transmission rate is 0.5 Mbps and target is the node 9. The node 1 and 3 each have ten adaptive TCP flows of which each window size is 11 and target is the node 9. The node 4 has forty non-adaptive UDP flows of which each transmission rate is 0.5 Mbps and target is the node 9. The node 4 results in congestion at link 7-8 and 8-9 by increasing its transmission rate every 1 seconds. The node 5 has ten adaptive TCP flows of which each window size is 15 and target is the node 10. We refer to the traffic generated by the node 5 as background traffic because its target is different from that of other source nodes.

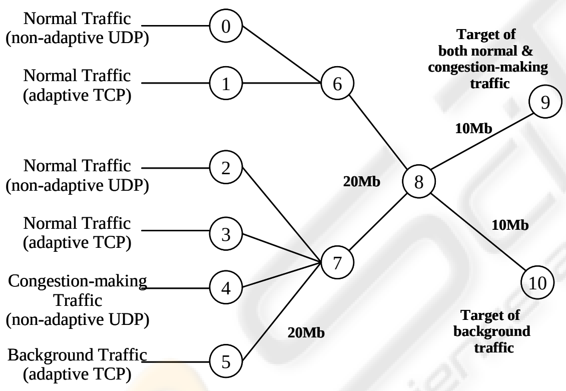
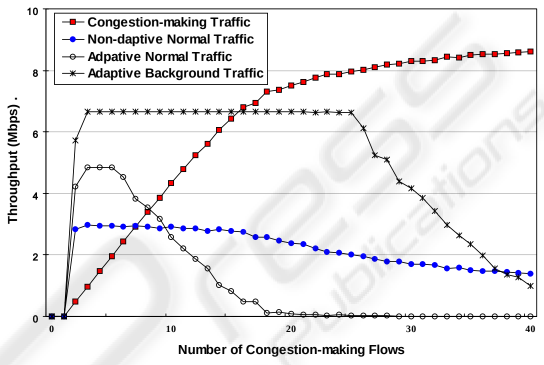

Normal Traffic (non-adaptive UDP)
Normal Traffic (adaptive TCP)
Normal Traffic (non-adaptive UDP)
Normal Traffic (adaptive TCP)
Congestion-making Traffic (non-adaptive UDP)
Background Traffic (adaptive TCP)
0
1
2
3
4
5
6
7
8
9
10
20Mb
20Mb
10Mb
10Mb
Target of both normal & congestion-making traffic
Target of background traffic

Figure 10: Network topology for simulation: link 7-8 and 8-9 are congested.

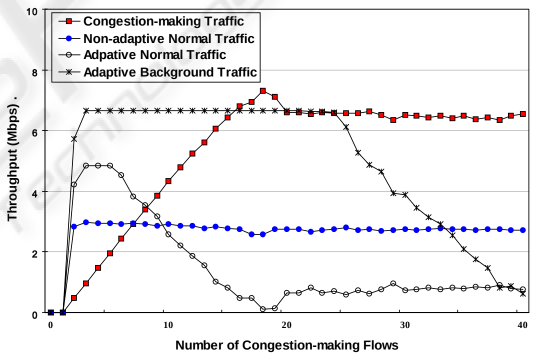
Fig. 11-(a),(b),(c),(d) show the simulation results of RED, ACC, PushBack (R. Mahajan et al, 2002), and CTC scheme, respectively. In RED, the congestion-making traffic consumes most of link bandwidth. As a result, both normal and background traffic are not good in throughput. ACC is better than RED in performance. ACC can protect the non-adaptive normal traffic. But, it fails to protect both the adaptive normal traffic and the background traffic. Pushback protects the background traffic as well as the non-adaptive normal traffic. But, Pushback fails to protect the adaptive normal traffic. This is because the adaptive normal traffic decreases its transmission when congestion occurs.

CTC protects all normal and background traffic as shown in Fig. 11-(d). CTC provides almost the full bandwidth that the normal users requested. This is because network congestion is localized to only congestion-making traffic without having any effect on normal traffic.

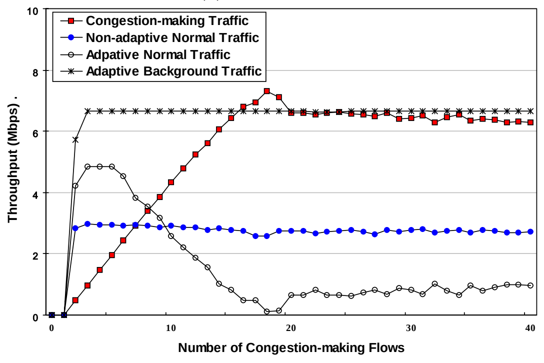
(a) RED



(B) ACC



(c) PushBack



(d) CTC proposed in this paper

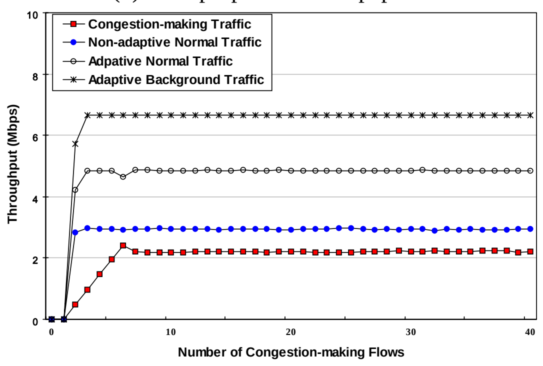
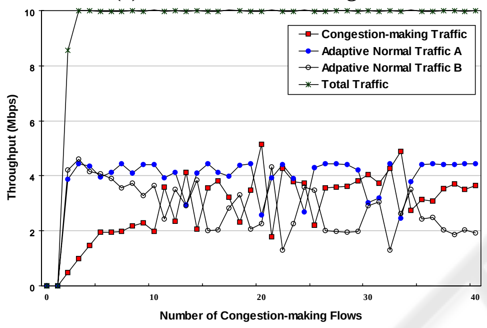


Figure 11: Throughput of adaptive TCP and non-adaptive UDP traffic under congestion

(a) CTC without marking



(b) CTC with marking

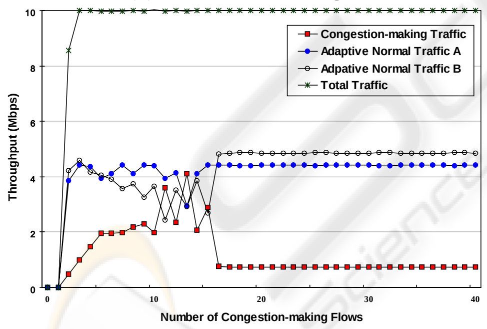


Figure 12: Non-marking vs Marking

CM-marking mechanism proposed in this paper is very useful when multiple congestions occur. Fig. 12-(a) and (b) show the simulation results of CTC without marking and CTC with marking, respectively. In this simulation, all normal sources generate only adaptive TCP traffic. As shown in Fig. 12, CTC with marking makes more exact determination because downstream node gets information on which one is congestion-making STT from upstream node

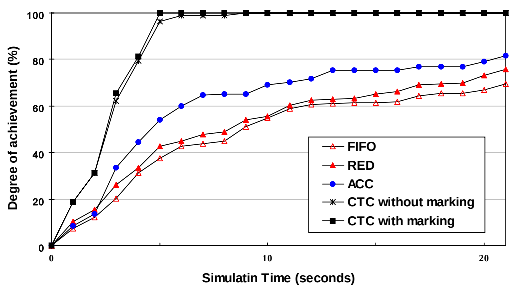


Figure 13: Time taken to finish a Web service under congestion

To compare CTC with the existing schemes in Web-like traffic environment, we rewrite link bandwidth in network topology shown in Fig. 10. Each uplink bandwidth between 6-8, between 7-8, and between 8-9 is set to 1 Mbps. In this simulation, the node 9 is a web server. Fig. 13 shows the time taken to finish a Web service under congestion. As shown Fig. 13, CTC is better than any other schemes.

5 RELATED STUDY

It is said that DDoS attack strategy rests on the followings (X. Geng et al, 2000):
(1) Using the Internet's insecure channels
(2) Hiding the attacker's identity
(3) Generating huge traffic volume

The first problem can be mitigated by installing intrusion detection system and firewall on customer networks, and also virus scan program and personal firewall on each user's PC.

To solve the second problem, there have been proposed a few approaches: ingress filtering (P. Ferguson et al, 2000), uRPF (Cisco, 2001), packet marking (S. Savage et al, 2001) and ICMP Traceback (S. Bellovin et al, 2001). Ingress filtering employs the scheme that boundary router filters all traffic coming from the customer that has a source address of something other than the addresses that have been assigned to the customer. uRPF discards faked packet by accepting only packet from interface if and only if the forwarding table entry for the source IP address matches the ingress interface. Packet marking scheme is one that probabilistically marks packets with partial path information as they arrive at routers in order that the victim may reconstruct the entire path. ICMP Traceback scheme uses a new ICMP message, emitted randomly by routers along the transmission path of packet and sent to the destination. The destination can determine the traffic source and path by using the ICMP messages.

This paper addresses the third and last problem. The problem can be thought of as the typical queuing discipline problem in network router. The core of the queuing discipline problem is to determine which packets get transmitted and which packets get discarded. There have been proposed many queuing algorithms (S. Keshav, 1997) such as FIFO: First-In-First-Out, FQ (Fair Queuing), RED (Random Early Detection), and so on. Those queuing algorithms cannot be used as a solution for the problem. For example, RED has a merit that the more packets sent by a flow, the higher the chance that its packets will be selected for dropping. But, RED also has a disadvantage that the more increase the volume of malicious user's traffic, the higher the probability that legitimate user's packet will be dropped because DDoS attacker can generate a huge volume of traffic. (F. Lau et al, 2000) recommended CBQ as the queuing algorithm that can protect legitimate user from DDoS attack. Using CBQ requires classification of traffic into each class. But, they didn't handle the problem.

There has been proposed static rate limit that blocks (or marks) packets exceeding a threshold (Cisco, 2000). This strategy is available only in DoS attack and also has a disadvantage that amount of packets during normal state should be first measured to fix the correct threshold value for limiting malicious traffic.

Yau has proposed max-min fair server-centric router throttle scheme (D.K.Y. Yau et al, 2002). The key idea is for a server under stress to install a router throttle (e.g. leaky-bucket) at selected upstream routers. The scheme can defeat DDoS traffic by controlling the router throttle. Mahajan has proposed a mechanism for detecting and controlling high bandwidth aggregates (R. Mahajan et al, 2002). They've researched recursive pushback of max-min fair rate limits starting from the target server to upstream routers. Both throttle and pushback mechanisms are likely to have a weak point in determining the threshold value for rate-limiting DDoS traffic and in requiring a new protocol for communication between victim and routers.

6 CONCLUSIONS

In this paper, we proposed a strong congestion-making traffic control scheme for preventing malicious or selfish user from congesting networks. Its key idea is to drop only packets corresponding to congestion-making traffic when network congestion occurs by providing congestion-making traffic with worse service (i.e., worse priority queue) than the normal traffic. We simulated the proposed scheme and the existing schemes to evaluate the performance of each scheme. The simulation results demonstrate that the proposed scheme is better than or almost same as the existing schemes in performance.

Even if our scheme is able to control congestion-making traffic effectively, we still need more research in analysing the attack traffic of malicious user in order to detect real congestion-making traffic generated by malicious user. We introduced IDA (Intrusion Detection Agent) in this paper. We think IDA will play an important role in defeating various kinds of attacks such as virus and worm, needlessly to say DoS attacks
Our future work is to implement and evaluate our scheme on real networks.

REFERENCES

S. Floyd, “TCP and explicit congestion notification,” ACM Computer Communication Review, vol. 24, no. 5, pp. 8.23, October 1994

K. J. Houle and G. M. Weaver. "Trends in Denial of Service Attack Technology," The fall 2001 NANOG meeting, Oct. 2001

X. Geng and A. B. Whinston, "Defeating Distributed Denial of Service Attacks", IT Pro, July-August 2000, pp 36-41

Cisco, "Strategies to Protect Against Distributed Denial of Service (DDoS) Attacks," white paper, http://www.cisco.com/…/newsflash.html, Feb. 2000.

R. Mahajan, S. M. Bellovin, S. Floyd, and et al., "Controlling High Bandwidth Aggregates in the Network," ACM SIGCOMM Computer Communications Review, Vol. 32, No. 3, pp. 62-73, July 2002.

P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827, May 2000.

Cisco, "Unicast Reverse Path Forwarding (uRPF) Enhancements for the ISP-ISP Edge", http://www. cisco.com/…/uRPF_Enhancement.pdf, Feb. 2001.

T. Li and Y. Rekhter "A Provider Architecture for Differentialted Services and Traffic Engineering (PASTE)". RFC 2430. October 1998.

K. Nichols,S. Blake, F. Baker and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474

F. Baker, W. Weiss and J. Wroclawski, "Assured Forwarding PHB Group," RFC 2597

V. Jacobson, K. Nichols, K. Poduri, "An Expedited Forwarding PHB," RFC 2598

UCB/LBNL/VINT, "ns Notes and Documentation," http://www.isi.edu/nsnam/ns.

S. Savage, A. Karlin and T. Anderson, "Network Support for IP Traceback," IEEE/ACM Transactions on Networking, Vol. 9, No. 3, June 2001, pp. 226-237

S. Bellovin, M. Leech, and T. Taylor, "ICMP Traceback Messages," Internet draft, Oct. 2001.

S. Keshav, "An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network", Addison Wesley, 1997.

F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic, "Distributed Denial of Service Attacks," IEEE International Conference on Systems, Man, and Cybernetics, 2000.

D.K.Y. Yau, J.C.S. Lui, and Feng Liang, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," Tenth IEEE International Workshop on Quality of Service, pp.35 - 44, May 2002.