Network Delineate Safeguard for Surplus Control

ANirmala[*], C.Kumuthini[**], V.Sridevi[**]

[*] Assistant Professor, Department of Computer Applications, Dr.N.G.P Arts and Science College, Coimbatore-48.
[**] Assistant Professor Dept. of Computer Applications, Dr.N.G.P Arts and Science College, Coimbatore-48.
[***] Assistant Professor Dept. of Computer Applications, Dr.N.G.P Arts and Science College, Coimbatore-48.

Abstract- The end-to-end nature of Internet congestion control is an important factor in its scalability and robustness. However, end-to-end congestion control algorithms alone are incapable of preventing the congestion collapse and unfair bandwidth allocations created by applications that are unresponsive to network congestion. To address this flaw, we propose and investigate a novel congestion avoidance mechanism called *Network Border Patrol* (NBP).

NBP relies on the exchange of feedback between routers at the borders of a network in order to detect and restrict unresponsive traffic flows before they enter the network. An enhanced core-stateless fair queueing mechanism is proposed in order to provide fair bandwidth allocations among competing flows. NBP is compliant with the Internet philosophy of pushing complexity toward the edges of the network whenever possible. Simulation results show that NBP effectively eliminates congestion collapse that, when combined with fair queueing,

NBP achieves approximately max-min fair bandwidth allocations for competing network flows.

I. INTRODUCTION

The essential philosophy behind the Internet is expressed by the scalability argument: no protocol, algorithm or service should be introduced into the Internet if it does not scale well. A key corollary to the scalability argument is the end-to-end argument: to maintain scalability, algorithmic complexity should be pushed to the edges of the network whenever possible. Perhaps the best example of the Internet philosophy is TCP congestion control, which is achieved primarily through algorithms implemented at end systems. Unfortunately, TCP congestion control also illustrates some of the shortcomings of the end-to-end argument.

A number of rate control algorithms have been proposed that are able to prevent congestion collapse and provide global max-min fairness to competing flows. These algorithms (e.g., ERICA, ERICA+) are designed for the ATM Available Bit Rate (ABR) service and require all network switches to compute fair allocations of bandwidth among competing connections. However, these algorithms are not easily adaptable to the current Internet, because they violate the Internet design philosophy of keeping router implementations simple and pushing complexity to the edges of the network.

Floyd and Fall have approached the problem of congestion collapse by proposing low-complexity router mechanisms that promote the use of adaptive or "TCP-friendly" end-to-end congestion control. Their suggested approach requires selected gateway routers to monitor high-bandwidth flows in order to determine whether they are responsive to congestion. Flows that are determined to be unresponsive are penalized by a higher packet discarding rate at the gateway router. A limitation of this approach is that the procedures currently available to identify unresponsive flows are not always successful. Hence we try to introduce and investigate a new Internet traffic control mechanism called the Protocol Encombrement Éviter which prevents congestion collapse by patrolling the network's borders, ensuring that packets do not enter the network at a rate greater than they are able to leave it. Associative Functional requirement Receives the packets on basis of frequency of the destination port.

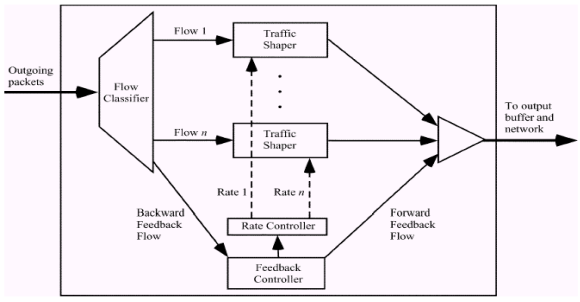
II. ARCHITECTURAL COMPONENTS

The only components of the network that require modification by NBP are edge routers; the input ports of egress routers must be modified to perform per-flow monitoring of bit rates, and the output ports of ingress routers must be modified to perform per-flow rate control. In addition, both the ingress and the egress routers must be modified to exchange and handle NBP feedback packets.

The input ports of egress routers are enhanced in NBP. Figure3.1 illustrates the architecture of an egress router's input port. Data packets sent by ingress routers arrive at the input port of the egress router and are first classified by flow. Flow classification is performed by ingress routers on every arriving packet based upon a flow classification policy.

An example flow classification policy is to examine the packet's source and destination *network addresses*, and to aggregate all packets arriving on an ingress router and destined to the same egress router into the same NBP flow (i.e., a macro-flow).

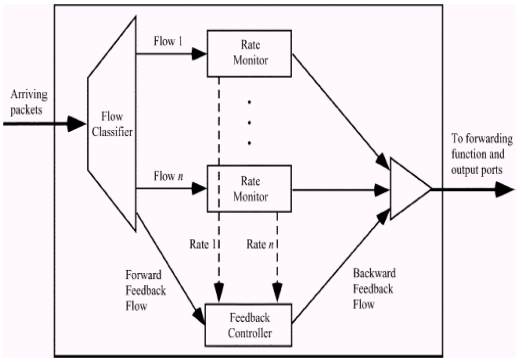
Figure 3.2: Input port of Egress router



After classifying packets into flows, each flow's bit rate is then rate monitored using a rate estimation algorithm such as the Time Sliding Window (TSW) algorithm. These rates are collected by a feedback controller, which returns them in

backward feedback packets to an ingress router whenever a forward feedback packet arrives from that ingress router.
The output ports of ingress routers are also enhanced in NBP. Each output port contains a flow classifier; per-flow traffic shapers (e.g., leaky buckets), a feedback controller, and a rate controller (Figure 3.2).

The flow classifier classifies packets into flows, and the traffic shapers limit the rates at which packets from individual flows enter the network. The feedback controller receives backward feedback packets returning from egress routers and passes their contents to the rate controller. It also generates forward feedback packets that are transmitted to the network's egress routers. To prevent congestion collapse, the rate controller adjusts traffic shaper parameters according to a TCP-like rate-control algorithm, and the rate-control algorithm used in NBP is described later in this section. Figure 3.3: Output port of Ingress Router



Algorithms used:

A set of algorithms have been implemented in order to design the core stateless protocol PEE. These algorithms have been listed below:
Rate Control Algorithm
Leaky Bucket Algorithm
Feedback Control
Time Sliding Window Algorithm

The algorithms have been dealt in detail in the following sections.
3.4.1 Rate Control Algorithm:

Figure 3.4: Rate Control Algorithm The PEE rate-control algorithm regulates the rate at which each flow is allowed to enter the network. Its primary goal is to converge on a set of per-flow transmission rates (hereinafter called *ingress rates*) that prevents congestion collapse due to undelivered packets. It also attempts to lead the network to a state of maximum link utilization and low router buffer occupancies, and it does this in a manner that is similar to TCP.

In the PEE rate-control algorithm, shown in Figure 3.4, a flow may be in one of two phases, *slow start* or *congestion avoidance*, similar to the phases of TCP congestion control. The desirable stability characteristics of slow-start and congestion control algorithms have been proven in TCP congestion control, and PEE expects to benefit from their well-known stability features. In PEE, new flows entering the network start with the *slow-start* phase and proceed to the *congestion-avoidance* phase only after the flow has experienced incipient congestion.

The rate-control algorithm is invoked whenever a backward feedback packet arrives at an ingress router. Recall that backward feedback packets contain a timestamp and a list of flows arriving at the egress router from the ingress router as well as the monitored egress rates for each flow. Upon the arrival of a backward feedback packet, the algorithm calculates the current round-trip time (*currentRTT* in Fig. 6) between the edge routers and updates the base round-trip time (*e.base RTT*), if necessary.

The base round-trip time (*e.base RTT*) reflects the best-observed round-trip time between the two edge routers. The algorithm then calculates *deltaRTT*, which is the difference between the current round-trip time (*currentRTT*) and the base round-trip time (*e.baseRTT*). A *deltaRTT* value greater than zero indicates that packets are requiring a longer time to traverse the network than they once did, and this can only be due to the buffering of packets within the network.

PEE's rate-control algorithm decides that a flow is experiencing incipient congestion whenever it estimates that the network has buffered the *equivalent* of more than one of the flow's packets at each router hop. To do this, the algorithm first computes the product of the flow's ingress rate (*f.ingressRate*) and *deltaRTT* (i.e., *f.ingressRate deltaRTT*). This value provides an estimate of the amount of the flow's data that is buffered somewhere in the network. If this amount (i.e., *f.ingressRate deltaRTT*) is greater than the number of router hops between the ingress and the egress routers (*e.hopcount*) multiplied by the size of the largest possible packet (MSS) (i.e., MSS *e.hopcount*), then the flow is considered to be experiencing incipient congestion.

The rationale for determining incipient congestion in this manner is to maintain both high link utilization and low queuing delay. Ensuring there is always at least one packet buffered for transmission on a network link is the simplest way to achieve full utilization of the link, and deciding that congestion exists when more than one packet is buffered at the link keeps queuing delays low.

Therefore, PEE's rate-control algorithm allows the "equivalent" of *e.hopcount* packets to be buffered in flow's path before it reacts to congestion by monitoring deltaRTT.1 a similar approach is used in the DEC bit congestion-avoidance mechanism. Furthermore, the approach used by PEE's rate control algorithm to detect congestion, by estimating whether the network has buffered the equivalent of more than one of the flow's packets at each router hop, has the advantage that, when congestion occurs, flows with higher ingress rates detect congestion first.

This is because the condition *f.ingressRate deltaRTT* MSS *e.hopcount* fails first for flows with a large ingress rate, detecting that the path is congested due to ingress flow.
When the rate-control algorithm determines that a flow is not experiencing congestion, it increases the flow's ingress rate. If the flow is in the slow-start phase, its ingress rate is doubled for each round-trip time that has elapsed since the last backward feedback packet arrived (*f.ingress).*

The estimated number of round-trip times since the last feedback packet arrived is denoted as *RTTs Elapsed*. Doubling the ingress rate during slow start allows a new flow to rapidly capture available bandwidth when the network is underutilized.

If, on the other hand, the flow is in the congestion-avoidance phase, then its ingress rate is conservatively incremented by one *rate Quantum* value for each round trip that has elapsed since the last backward feedback packet arrived (*f.ingressrate rate Quantum RTTsElapsed*). This is done to avoid the creation of congestion. The rate quantum is computed as the maximum segment size divided by the current round-trip time between the edge routers. This results in rate growth behavior that is similar to TCP in its congestion-avoidance phase.

Furthermore, the rate quantum is not allowed to exceed the flow's current egress rate divided by a constant quantum factor (QF). This guarantees that rate increments are not excessively large when the round-trip time is small. When the rate-control algorithm determines that a flow is experiencing incipient congestion, it reduces the flow's ingress rate.

If a flow is in the slow-start phase, it enters the congestion-avoidance phase. If a flow is already in the congestion-avoidance phase, its ingress rate is reduced to the flow's egress rate decremented by a constant value. In other words, an observation of incipient congestion forces the ingress router to send the flow's packets into the network at a rate slightly lower than the rate at which they are leaving the network.

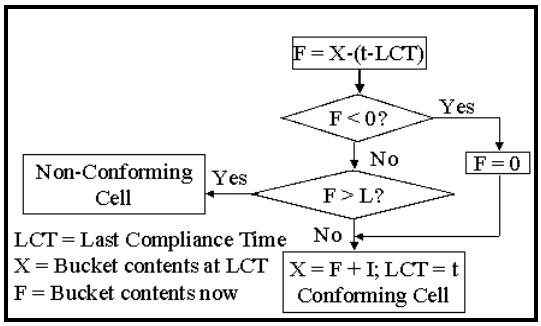
PEE's rate-control algorithm is designed to have minimum impact on TCP flows. The rate at which PEE regulates each flow (*f.ingressRate*) is primarily a function of the round-trip time between the flow's ingress and egress routers (*currentRTT*). In PEE, the initial ingress rate for a new flow is set to be MSS/*e.baseRTT*, following TCP's initial rate of one segment per round-trip time.

PEE's *currentRTT* is always smaller than TCP's end-to-end round-trip time (as the distance between ingress and egress routers, i.e., the *currentRTT* in PEE, is shorter than the end-to-end distance, i.e., TCP's round-trip time). As a result, *f.ingressRate* is normally larger than TCP's transmission rate when the network is not congested, since the TCP transmission window increases at a rate slower than PEE's *f.ingressRate* increases. Therefore, PEE normally does not regulate TCP flows. However, when congestion occurs, PEE reacts first by reducing *f.ingressRate* and, therefore, reducing the rate at which TCP packets are allowed to enter the network. TCP eventually detects the congestion (either by losing packets or due to longer round-trip times) and then promptly reduces its transmission rate. From this time point on, *f.ingressRate* becomes greater than TCP's transmission rate, and therefore, PEE's congestion control does not regulate TCP sources until congestion happens again.

Leaky Bucket Algorithm:

The leaky bucket algorithm is used to regulate the traffic flow from the input port to the output port. We assume leaky bucket as a bucket with a small hole at the bottom. Hence any packet that enters the bucket at any rate must go out of the bucket at a controlled rate from the hole at the bottom. Also we assume that the limit of the bucket is infinity. Hence there is no case of bucket getting filled and the packets getting lost due to the limit of the bucket.

The leaky bucket algorithm is described as a flowchart in Figure



Feedback Control Algorithm:

The feedback control algorithm in NBP determines how and when feedback packets are exchanged between edge routers. Feedback packets take the form of ICMP packets and are necessary in NBP for three reasons. First, forward feedback packets allow egress routers to discover which ingress routers are acting as sources for each of the flows they are monitoring.
Second, backward feedback packets allow egress routers to communicate per-flow bit rates to ingress routers. Third, forward and backward feedback packets allow ingress routers to detect incipient network congestion by monitoring edge-to-edge round-trip times.

Time Sliding Window algorithm:

It is used for rate monitoring. TSW estimates the sending rate upon each packet arrival, and decays, or forgets the past history over time. TSW maintains three state variables in the hash structure: Win_length, which is measured in units of time, Avg_rate, the rate estimate upon each packet arrival, and T_front, which is the time of last packet arrival. TSW is used to estimate the rate upon each packet arrival, so state variables Avg_rate and T_front are updated each time a packet arrives. Win_length is pre-configured.

III. CONCLUSION

In this paper, we have presented a novel congestion avoidance mechanism for the Internet called Network Border

Patrol and an enhanced core-stateless fair queuing mechanism. Unlike existing Internet congestion control approaches, which rely solely on end-to-end control, NBP is able to prevent congestion collapse from undelivered packets. It does this by ensuring at the border of the network that each flow's packets do not enter the network faster than they are able to leave it. NBP requires no modifications to core routers nor to end systems. Only edge routers are enhanced so that they can perform the requisite per-flow monitoring, per-flow rate control and feedback exchange operations.

Extensive simulation results provided in this paper show that NBP successfully prevents congestion collapse from undelivered packets. They also show that, while NBP is unable to eliminate unfairness on its own, it is able to achieve approximate global max-min fairness for competing network flows when combined with ECSFQ, they approximate global max-min fairness in a completely core-stateless fashion.

REFERENCES

[1] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, August 1999, To appear.

[2] J. Nagle, "Congestion control in IP/TCP internetworks," Request for Comments 896, Internet Engineering Task Force, Jan. 1984.

[3] Van Jacobson, "Congestion avoidance and control," *ACM Computer Communications Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988, Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[4] "Real Broadcast Network White Paper," White paper, RealNetworks Inc., January 1999, http://www.real.com/solutions/rbn/whitepaper.html.

[5] "RealVideo Technical White Paper," White paper, RealNetworks Inc., January 1999, http://www.real.com/devzone/library/whitepapers/overview.html.

[6] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proc. of ACM SIGCOMM*, September 1998, pp. 303–314.

[7] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *Proc. of ACM SIGCOMM*, September 1989, pp. 1–12.

[8] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control – the Single Node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[9] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed

[10] Networks," in *Proc. of ACM SIGCOMM*, September 1998, pp. 118–130.

AUTHORS

First Author – ANirmala, Assistant Professor, Department of Computer Applications, Dr.N.G.P Arts and Science College, Coimbatore-48.

Second Author – C.Kumuthini, Assistant Professor Dept. of Computer Applications, Dr.N.G.P Arts and Science College, Coimbatore-48.

Third Author – V.Sridevi, Assistant Professor Dept. of Computer Applications, Dr.N.G.P Arts and Science College, Coimbatore-48.