Network Border Patrol: Preventing Congestion Collapse and Promoting Fairness in the Internet *

Célio Albuquerque†, Brett J. Vickers‡ and Tatsuya Suda†

† Dept. of Information and Computer Science
University of California, Irvine
{celio,suda}@ics.uci.edu

‡ Dept. of Computer Science
Rutgers University
bvickers@cs.rutgers.edu

Abstract

The Internet's excellent scalability and robustness result in part from the end-to-end nature of Internet congestion control. End-to-end congestion control algorithms alone, however, are unable to prevent the congestion collapse and unfairness created by applications that are unresponsive to network congestion. To address these maladies, we propose and investigate a novel congestion avoidance mechanism called *Network Border Patrol* (NBP). NBP entails 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, thereby preventing congestion within the network. Moreover, NBP is complemented with the proposed enhanced core-stateless fair queueing (ECSFQ) mechanism, which provides fair bandwidth allocations to competing flows. Both NBP and ECSFQ are 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 and that, when combined with ECSFQ, approximately max-min fair bandwidth allocations can be achieved for competing flows.

Keywords: Internet, congestion control, congestion collapse, max-min fairness, end-to-end argument, core-stateless mechanisms, border control

*This work was supported by the National Science Foundation through grants NCR-9628109, ANI-0083074, and ANI-9903427, by DARPA through Grant MDA972-99-1-0007, by AFOSR through Grant MURI F49620-00-1-0330, and by grants from the University of California MICRO Program, Hitachi, Hitachi America, Standard Microsystems Corporation, Canon USA, Novell, Tokyo Electric Power Company, Nippon Telegraph and Telephone Corporation (NTT), NTT Docomo, Fujitsu, Nippon Steel Information and Communication Systems Incorporated (ENICOM), Canon and Matsushita Electric Industrial company, and Fundação CAPES/Brazil.

1 Introduction

The fundamental philosophy behind the Internet is expressed by the scalability argument: no protocol, mechanism 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 implemented primarily through algorithms operating at end systems. Unfortunately, TCP congestion control also illustrates some of the shortcomings of the end-to-end argument. As a result of its strict adherence to end-to-end congestion control, the current Internet suffers from two maladies: congestion collapse from undelivered packets, and unfair allocations of bandwidth between competing traffic flows.

The first malady—congestion collapse from undelivered packets—arises when bandwidth is continually consumed by packets that are dropped before reaching their ultimate destinations [1]. John Nagle assigned the term "congestion collapse" in 1984 to describe a network that remains in a stable congested state [2]. At that time, the primary cause of congestion collapse was inefficient use of retransmission timers by TCP sources, which led to the unnecessary retransmission of delayed packets. This problem was corrected with more recent implementations of TCP [3]. Recently, however, a potentially more serious cause of congestion collapse has become increasingly common. Network applications are now frequently written to use transport protocols, such as UDP, which are oblivious to congestion and make no attempt to reduce packet transmission rates when packets are discarded by the network [4]. In fact, during periods of congestion some applications actually *increase* their transmission rates by introducing redundancy in the transmitted data in order to become less sensitive to packet losses [5]. The Internet presently has no effective way to regulate such applications.

The second malady—unfair bandwidth allocation to competing network flows—arises in the Internet for a variety of reasons, one of which is the existence of applications that do not respond properly to congestion. Adaptive applications (e.g., TCP-based applications) that respond to congestion by rapidly reducing their transmission rates are likely to receive unfairly small bandwidth allocations when competing with unresponsive applications. The Internet protocols themselves can also introduce unfairness. The TCP algorithm, for instance, inherently causes each TCP flow to receive a bandwidth that is inversely proportional to its round trip time [6]. Hence, TCP connections with short round trip times may receive unfairly large allocations of network bandwidth when compared to connections with longer round trip times.

The impact of emerging streaming media traffic on traditional data traffic is of growing concern in the Internet

community. Streaming media traffic is unresponsive to the congestion in a network, and it can aggravate congestion collapse and unfair bandwidth allocation. Recently, various researchers have documented and studied the problems of unfairness and congestion collapse due to unresponsive traffic, such as streaming media traffic [7, 8, 9, 10, 11, 12]. This concern regarding the negative impact that streaming media traffic may bring has also been expressed in the industry and in the IETF, and in August 1999, the New York Times reported concern of ISPs with multimedia transmissions driving the network to a gridlock [13].

To address the maladies of congestion collapse and unfairness, we introduce and investigate a novel Internet traffic control protocol called *Network Border Patrol*. The basic principle of Network Border Patrol (NBP) is to compare, at the borders of a network, the rates at which packets from each application flow are entering and leaving the network. If a flow's packets are entering the network faster than they are leaving it, then the network is likely buffering or, worse yet, discarding the flow's packets. In other words, the network is receiving more packets than is capable of handling. NBP prevents this scenario by "patrolling" the network's borders, ensuring that each flow's packets do not enter the network at a rate greater than they are able to leave the network. This patrolling prevents congestion collapse from undelivered packets, because unresponsive flow's otherwise undeliverable packets never enter the network in the first place.

In order to achieve fair bandwidth allocations among competing flows, Network Border Patrol may be used in conjunction with an appropriate fair queueing mechanism. Weighted fair queuing (WFQ) [14, 15] is an example of one such mechanism. Unfortunately, WFQ imposes significant complexity on interior network routers by requiring them to maintain per-flow state and perform per-flow scheduling of packets. In this paper we propose an *Enhanced Core-Stateless Fair Queuing (ECSFQ) mechanism*, in order to achieve some of the advantages of WFQ without most of its complexity, and we use the ECSFQ mechanism to improve NBP's fairness.

Although NBP is capable of preventing congestion collapse and improving the fairness of bandwidth allocations, these improvements do not come for free. NBP solves these problems at the expense of some additional network complexity, since routers at the border of the network are expected to monitor and control the rates of individual flows in NBP. NBP also introduces added communication overhead, since in order for an edge router to know the rate at which its packets are leaving the network, it must exchange feedback with other edge routers. Unlike some existing approaches trying to solve congestion collapse, however, NBP's added complexity is isolated to edge routers; routers within the core of the network do not participate in the prevention of congestion collapse. Moreover, end systems operate in total ignorance of the fact that NBP is implemented in the network, so no changes to transport protocols are necessary at end systems.

The remainder of this paper is organized as follows. In Section 2 we describe why existing mechanisms are not

effective in preventing congestion collapse or providing fair bandwidth allocations in the presence of unresponsive flows. In section 3 we describe the architectural components of Network Border Patrol in further detail and present the feedback and rate control algorithms used by NBP edge routers to prevent congestion collapse. In Section 4 we explain the enhanced core-stateless fair queueing mechanism and illustrate the advantages of providing lower queueing delays to flows transmitting at lower rates. In section 5, we present simulations results, showing the ability of NBP to avoid congestion collapse and provide fair bandwidth allocations to competing flows. In section 6, we discuss several implementation issues that must be addressed in order to make deployment of NBP feasible in the Internet. Finally, in section 7 we provide some concluding remarks.

2 Related Work

The maladies of congestion collapse from undelivered packets and of unfair bandwidth allocations have not gone unrecognized. Some have argued that there are social incentives for multimedia applications to be friendly to the network, since an application would not want to be held responsible for throughput degradation in the Internet. Nevertheless, malicious denial-of-service attacks using unresponsive UDP flows are becoming disturbingly frequent in the Internet, and they are an example that the Internet cannot rely solely on social incentives to control congestion or to operate fairly.

Some have argued that congestion collapse and unfairness can be mitigated through the use of improved packet scheduling [16] or queue management [17] mechanisms in network routers. For instance, per-flow packet scheduling mechanisms such as Weighted Fair Queueing (WFQ) [14, 15] attempt to offer fair allocations of bandwidth to flows contending for the same link. So do Core-Stateless Fair Queueing (CSFQ) [18], Rainbow Fair Queueing [19] and CHOKe [20], which are cost-effective approximations of WFQ that do not require core routers to maintain per-flow state. Active queue management mechanisms like Fair Random Early Detection (FRED) [21] also attempt to limit unresponsive flows by specifically discarding packets from flows that are using more than their fair share of a link's bandwidth. All of these mechanisms reduce the likelihood of unfairness and congestion collapse in the Internet, but they do not eliminate them, and many of them are much more complex and expensive to implement than simple FIFO queueing.

For illustration, consider the example shown in Figure 1. In this example, two unresponsive flows (flow A and flow B) compete for bandwidth in a network containing two bottleneck links (R_1-R_2 and R_2-S_4) arbitrated by a fair queueing mechanism at routers R_1 and R_2. At the first bottleneck link (R_1-R_2), fair queueing at router R_1 ensures that each flow receives half of the link's available bandwidth (750 kbps). On the second bottleneck link (R_2-S_4), much of

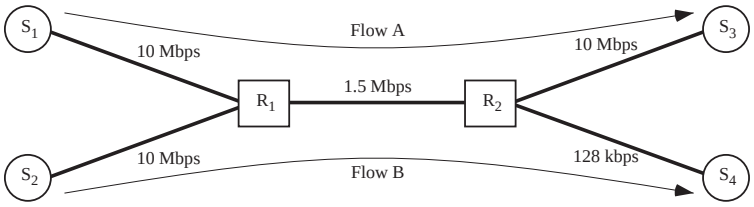


Figure 1: Example of a network which experiences congestion collapse

the traffic from flow B is discarded due to the link's limited capacity (128 kbps). Hence, flow A achieves a throughput of 750 kbps, and flow B achieves a throughput of 128 kbps. Clearly, congestion collapse has occurred, because flow B's packets, which are ultimately discarded on the second bottleneck link (R_2-S_4), limit the throughput of flow A across the first bottleneck link (R_1-R_2). Furthermore, while both flows receive equal bandwidth allocations on the first bottleneck link, their allocations are not *globally max-min fair*. An allocation of bandwidth is said to be globally max-min fair if, at every link, all active flows not bottlenecked at another link are allocated a maximum, equal share of the link's remaining bandwidth [22]. A globally max-min fair allocation of bandwidth for the example shown in Figure 1 would have been 1.372 Mbps for flow A and 128 kbps for flow B.

The example discussed in the previous paragraph, which is a variant of an example presented by Floyd and Fall [1], illustrates the inability of local scheduling mechanisms, such as WFQ, to eliminate congestion collapse and achieve global max-min fairness and suggests the need for the assistance of additional network mechanisms.

Jain *et al.* have proposed several rate control algorithms [23] 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 tailorable 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.

Rangarajan and Acharya proposed a network border-based approach, which aims to prevent congestion collapse through early regulation of unresponsive flows (ERUF) [11]. Border routers rate control the input traffic, while core routers generate ICMP source quench messages when packet drops occur in order to advise sources and border routers to reduce their sending rates. While this approach may prevent congestion collapse, it does so only after packets have been dropped and the network is congested. This approach also lacks mechanisms to provide fair bandwidth allocations to competing network flows.

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 [1]. Their suggested approach

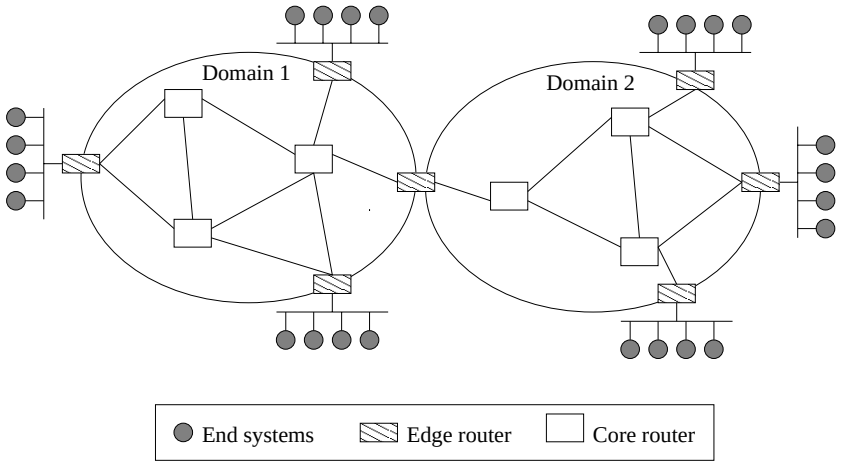


Figure 2: The core-stateless Internet architecture assumed by NBP

requires selected gateway routers to monitor high-bandwidth flows in order to determine whether they are responsive to congestion. Flows determined to be unresponsive to congestion 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 [18].

3 Network Border Patrol

Network Border Patrol is a network layer congestion avoidance protocol that is aligned with the "core-stateless" approach. The core-stateless approach, which has recently received a great deal of research attention [24, 18], allows routers on the borders (or edges) of a network to perform flow classification and maintain per-flow state but does not allow routers at the core of the network to do so. Figure 2 illustrates this architecture. As in other work on core-stateless approaches, we draw a further distinction between two types of edge routers. Depending on which flow it is operating on, an edge router may be viewed as an *ingress* or an *egress* router. An edge router operating on a flow passing into a network is called an ingress router, whereas an edge router operating on a flow passing out of a network is called an egress router. Note that a flow may pass through more than one egress (or ingress) router if the end-to-end path crosses multiple networks.

NBP prevents congestion collapse through a combination of per-flow rate monitoring at egress routers and per-flow rate control at ingress routers. Rate monitoring allows an egress router to determine how rapidly each flow's packets are leaving the network, whereas rate control allows an ingress router to police the rate at which each flow's packets enter the network. Linking these two functions together are the feedback packets exchanged between ingress and egress routers; ingress routers send egress routers *forward* feedback packets to inform them about the flows that are

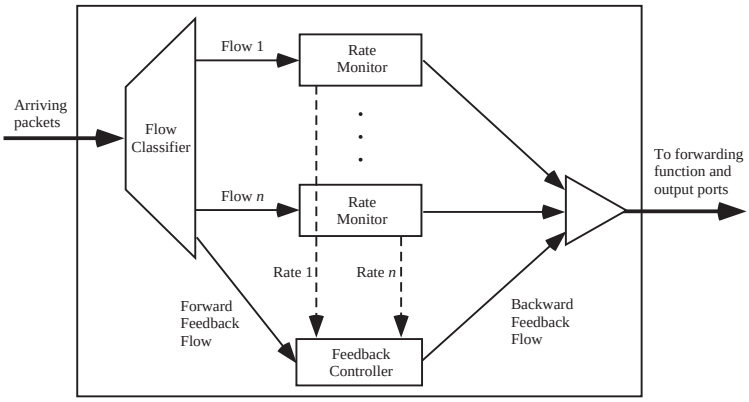


Figure 3: An input port of an NBP egress router

being rate controlled, and egress routers send ingress routers *backward* feedback packets to inform them about the rates at which each flow's packets are leaving the network. By matching the ingress rate and egress rate of each flow, NBP prevents congestion collapse within the network[1].

This section describes three important aspects of the NBP mechanism: (1) the architectural components, namely the modified edge routers, which must be present in the network, (2) the feedback control algorithm, which determines how and when information is exchanged between edge routers, and (3) the rate control algorithm, which uses the information carried in feedback packets to regulate flow transmission rates and thereby prevent congestion collapse in the network.

3.1 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. Figure 3 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 depending on the protocol used and size of the network. In the case of IPv6, flows may be classified by examining the packet header's flow label, whereas in the case of IPv4, it is done by examining the packet's source and destination addresses and port numbers. In small networks, flows may be classified

[1]One added side effect of NBP is that it can help prevent denial-of-service (DoS) attacks. Although NBP is not specifically designed to prevent DoS attacks, if the DoS attack relies on unresponsive UDP traffic to create congestion in the target network, NBP can be effective in preventing the attacks.

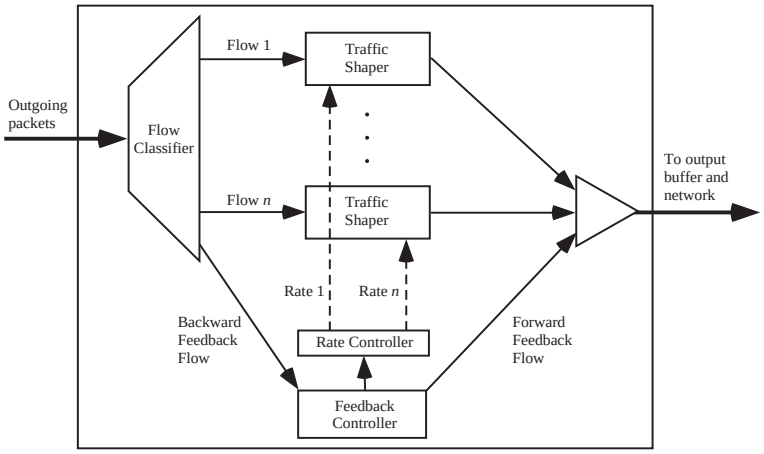


Figure 4: An output port of an NBP ingress router

according to the source and destination addresses (i.e., micro-flows). In large networks, flows should be classified in a more scalable fashion by examining the packet's source and destination *network addresses*, and by aggregating packets arriving on an ingress router and destined to the same egress router into the same flow (i.e., a macro-flow). 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) [25]. 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. See Figure 4. 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.

3.2 The 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.

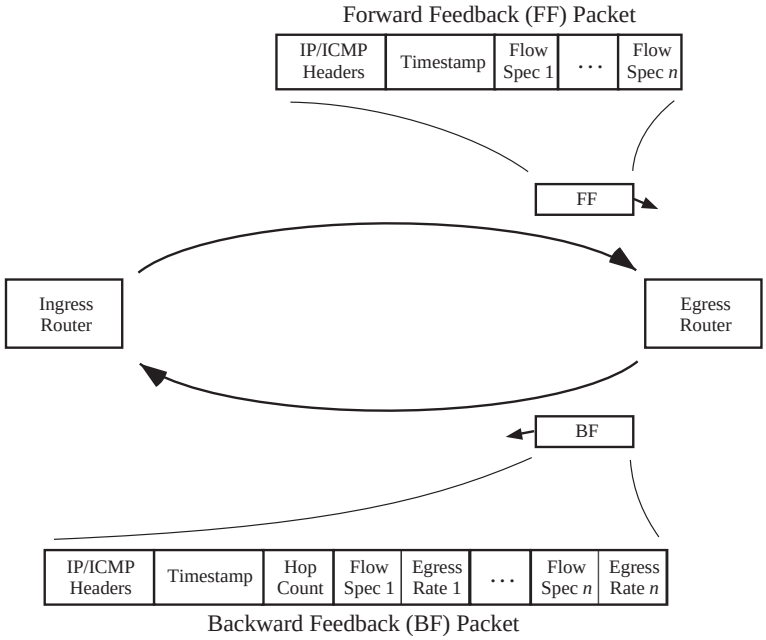


Figure 5: Forward and backward feedback packets exchanged by edge routers

The contents of feedback packets are shown in Figure 5. Contained within the forward feedback packet generated at an ingress router are a time stamp and a list of flow specifications for flows originating at the ingress router. The time stamp field is used to calculate the round trip time between two edge routers, and the list of flow specifications indicates to an egress router the identities of active flows originating at the ingress router. A flow specification is a value uniquely identifying a flow, assigned by the ingress router flow classifier. An ingress router adds a flow to its list of active flows whenever a packet from a new flow arrives; it removes a flow when the flow becomes inactive. In the event that the network's maximum transmission unit size is not sufficient to hold an entire list of flow specifications, multiple forward feedback packets are used.

When an egress router receives a forward feedback packet, it immediately generates a backward feedback packet and returns it to the ingress router. Contained within the backward feedback packet are the forward feedback packet's original time stamp, a hop count, and a list of observed bit rates, called *egress rates*, collected by the egress router for each flow listed in the forward feedback packet. The hop count, which is used by the ingress router's rate control algorithm, indicates how many routers are in the path between the ingress and the egress router. The egress router determines the hop count by examining the time to live (TTL) field of arriving forward feedback packets. When the backward feedback packet arrives at the ingress router, its contents are passed to the ingress router's rate controller, which uses them to adjust the parameters of each flow's traffic shaper.

In order to determine how often to generate forward feedback packets, an ingress router keeps a byte transmission counter for each flow it monitors. Whenever a flow's byte transmission counter exceeds a threshold, denoted *NBP's*

```
on arrival of Backward Feedback packet p from egress router e
   currentRTT  = currentTime - p.timestamp;
   if (currentRTT < e.baseRTT)
       e.baseRTT = currentRTT;
   deltaRTT = currentRTT - e.baseRTT;
   RTTsElapsed = (currentTime - e.lastFeedbackTime) / currentRTT;
   e.lastFeedbackTime  = currentTime;
   for each flow f listed in p
       rateQuantum = min (MSS / currentRTT,  f.egressRate / QF);
       if  (f.phase == SLOW_START)
          if  (deltaRTT × f.ingressRate < MSS × e.hopcount)
              f.ingressRate = f.ingressRate × 2 ^ RTTsElapsed;
          else
             f.phase = CONGESTION_AVOIDANCE;
       if  (f.phase == CONGESTION_AVOIDANCE)
          if  (deltaRTT × f.ingressRate < MSS × e.hopcount)
              f.ingressRate = f.ingressRate + rateQuantum × RTTsElapsed;
          else
              f.ingressRate = f.egressRate - rateQuantum;
```

Figure 6: Pseudocode for ingress router rate control algorithm

transmission counter threshold (T_x), the ingress router generates and transmits a forward feedback packet to the flow's egress router, and resets the the byte transmission counters of all flows included in the feedback packet. Using a byte transmission counter for each flow ensures that forward feedback packets are generated more frequently when flows transmit at higher rates, thereby allowing ingress routers to respond more quickly to impending congestion collapse. To maintain a frequent flow of feedback between edge routers even when data transmission rates are low, ingress routers also generate forward feedback packets whenever a time-out interval, denoted τ_f, is exceeded.

A rough estimate of the amount of overhead that NBP feedback packets create is provided in the Appendix of this paper.

3.3 The Rate Control Algorithm

The NBP 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 NBP rate control algorithm, shown in Figure 6, 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 NBP expects to benefit from their

well-known stability features. In NBP, 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 Figure 6) between the edge routers and updates the base round trip time (*e.baseRTT*), if necessary. The base round trip time (*e.baseRTT*) 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.

NBP'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* $\times$ *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* $\times$ *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 $\times$ 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 queueing 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 queueing delays low.

Therefore, NBP's rate control algorithm allows the "equivalent" of *e.hopCount* packets to be buffered in flow *f*'s path before it reacts to congestion by monitoring deltaRTT [2]. A similar approach is used in the DECbit congestion avoidance mechanism [26]. Furthermore, the approach used by NBP'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 of , when congestion occurs, flows with higher ingress rates detect congestion first. This is because the condition *f.ingressRate* $\times$ *deltaRTT* ¡ *MSS* $\times$ *e.hopcount* fails first for flows *f* with a large ingress rate, detecting

[2]Notice that *deltaRTT* increases if packets from any flow are buffered somewhere on the path of flow *f*. When routing changes occur, *deltaRTT* may also change. In the current Internet routing changes occur in a time scale much larger than the reaction time of congestion control algorithms, and NBP requires *e.baseRTT* to be refreshed if routing changes within the network occur.

that the path is congested due to ingress flow *f*.

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* $\times 2^{RTTsElapsed}$). The estimated number of round trip times since the last feedback packet arrived is denoted as *RTTsElapsed*. 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 *rateQuantum* value for each round trip that has elapsed since the last backward feedback packet arrived (*f.ingressRate* + *rateQuantum* $\times$ *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 [3].

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.

NBP's rate control algorithm is designed to have minimum impact on TCP flows. The rate at which NBP regulates each flow (*f.ingressRate*) is primarily a function of the round trip time between the flow's ingress and egress routers (*currentRTT*). In NBP, 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. NBP'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 NBP, is shorter than end to end distance, i.e., TCP's RTT). As a result, *f.ingressRate* is normally larger than TCP's transmission rate when the network is not congested, since TCP transmission window increases at a rate slower than NBP's *f.ingressRate* increases. Therefore, NBP normally does not regulate TCP flows. However, when congestion occurs, NBP 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

[3]The rate quantum should be small enough to allow the ingress rate to converge. The rate quantum is the minimum between (*MSS/currentRTT*) and the (*f.egressRate/QF*). Through simulations, we recommend that values for *QF* are between 5 and 50. If *QF* is too large, the rate quantum becomes too small and the convergence time of the ingress rate may take too long. If *QF* is too small, the rate quantum may become too large depending on the RTT, and may lead the system to an unstable state.

congestion (either by loosing 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, NBP's congestion control do not regulate TCP sources until congestion happens again.

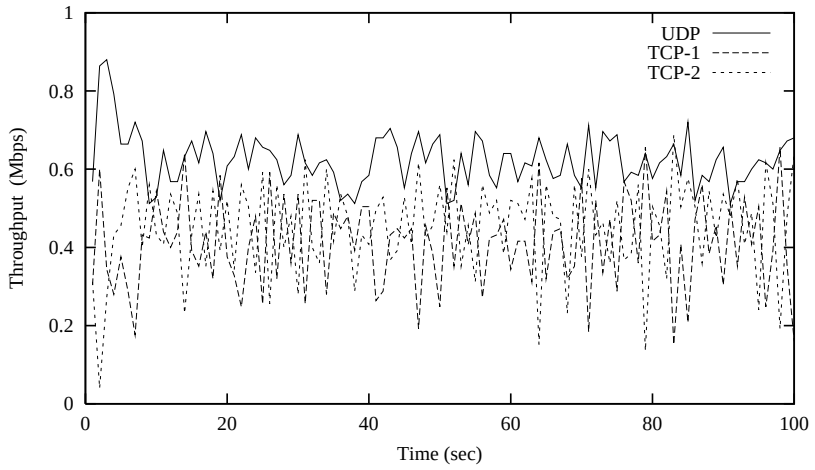
4 Adding Fairness to Network Border Patrol

Network Border Patrol's feedback control algorithm detects incipient congestion by monitoring the edge-to-edge round trip times. The rate control algorithm regulates the rate at which flows enter the network. The combination of both algorithms ensures that excessive flow traffic is not allowed to enter the network if incipient congestion is detected, thereby preventing congestion collapse. Although Network Border Patrol prevents congestion collapse, it does not guarantee that all flows are treated fairly when they compete for bottleneck links. To address this concern, we consider the interoperation of Network Border Patrol and various fair queueing mechanisms. We also introduce the *Enhanced Core-Stateless Fair Queueing mechanism*, in order to introduce fairness to NBP in a core-stateless fashion.

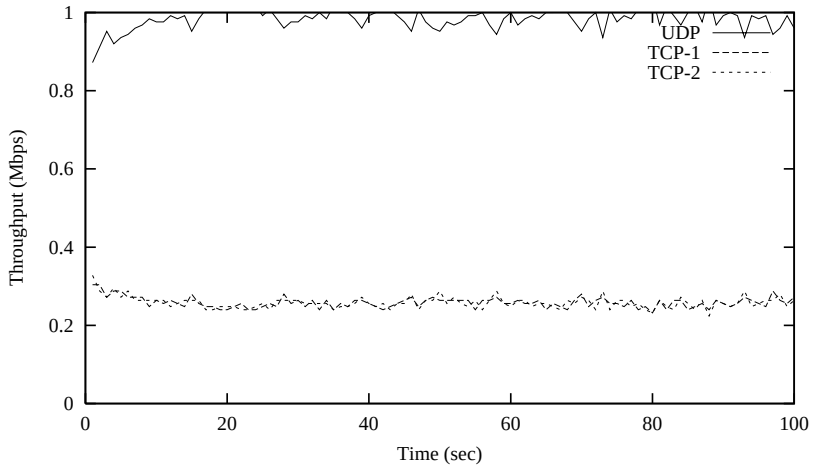
Fair bandwidth allocations can be achieved by using per-flow packet scheduling mechanisms such as fair queuing [14, 15]. As discussed in the related work section, fair queuing fairly allocates bandwidth to packet flows competing for a single link. However, in order to provide this benefit, it requires each link to maintain separate queues and state for each flow. This complexity overhead impedes the scalability of fair queuing, making it impractical for wide area networks in which a significantly large number of flows may be active at any one time.

Recognizing the scalability difficulties of fair queuing, several researchers have proposed more scalable "core-stateless" approximations of fair queuing, such as Core-Stateless Fair Queuing [18], Rainbow Fair Queuing [19] and CHOke [20]. The basic idea behind these mechanisms is that edge routers label packets entering the network with the state of the packets' flows, and core routers use the state recorded in the packets to decide whether to drop them or schedule them for transmission. These core stateless mechanisms are more scalable than fair queueing, because they limit per-flow operations and state maintenance to routers on the edges of a network.

Although existing core-stateless fair queuing mechanisms work well with most congestion control algorithms that rely on packet losses to indicate congestion [3, 2, 27], they do not work as well with congestion avoidance algorithms that prevent congestion before packet loss occurs. Examples of such congestion avoidance algorithms include TCP Vegas [28, 29], TCP with Explicit Congestion Notification [30] and Network Border Patrol. Two simulation experiments shown in Figures 7(a) and 7(b) illustrate this phenomenon. In both experiments, two TCP flows and a 1 Mbps constant bit rate UDP flow share a single 1.5 Mbps bottleneck link, as shown in Figure 1. We use CSFQ as a representative example of the core-stateless fairness mechanisms. In the first experiment, the TCP sources use the TCP Reno imple-



(a) CSFQ achieves approximately fair bandwidth allocations when TCP Reno sources are used.



(b) CSFQ fails to achieve fair bandwidth allocations when TCP Vegas sources are used.

Figure 7: CSFQ does not achieve fair bandwidth allocations when used with some congestion avoidance mechanisms

mentation, which relies on observations of packet loss to indicate congestion. As Figure 7(a) shows, the core-stateless mechanism provides approximately fair allocations to all three flows when used with algorithms that rely on packet losses to indicate congestion. In the second experiment, the TCP Reno sources are replaced by TCP Vegas sources, which rely on round trip time measurements to predict incipient congestion and keep buffer occupancies small. Here, as Figure 7(b) shows, the core-stateless mechanism fails to provide fair allocations of bandwidth to the TCP flows.

CSFQ fails when congestion avoidance algorithms that prevent packet loss are used, because it does not accurately approximate the delay characteristics of fair queuing. In fair queuing, flows transmitting at rates less than or equal to their fair share are guaranteed timely delivery of their packets since they do not share the same buffer as packets from other flows. In the core-stateless approximations of fair queuing, this is not the case, since they aggregate packets from

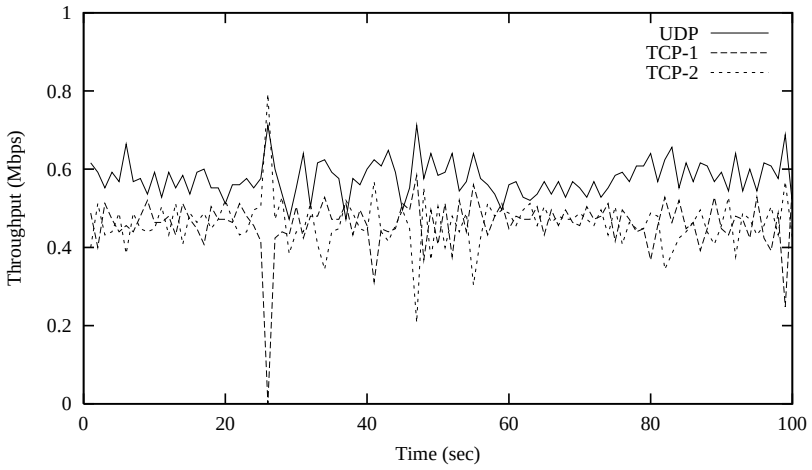


Figure 8: Enhanced CSFQ restores fairness when used with TCP Vegas

all flows into a single buffer and rely on packet discarding to balance the service of each flow. Hence, the existing core-stateless mechanisms are incompatible with congestion avoidance mechanisms that maintain small router buffers or rely on round trip time measurements to indicate incipient congestion.

In order to overcome the disadvantages of existing core-stateless fair queueing mechanisms, we propose a slightly modified version of CSFQ, hereinafter referred to as Enhanced CSFQ (ECSFQ). ECSFQ not only achieves the scalability of CSFQ, but at the same time it achieves fair bandwidth allocations when used with preventive congestion avoidance mechanisms like TCP Vegas and Network Border Patrol. The basic idea of ECSFQ is to introduce, in a core-stateless manner, an additional high priority buffer at each core router. The high priority buffer is used to hold packets from flows transmitting at rates less than their fair share, while the original buffer holds the remaining packets. Packets in the high priority buffer are served first and therefore experience short delays. Once a flow's rate meets or exceeds its fair share, the flow's packets enter the low priority buffer and its packets experience the same delays as packets from other existing flows transmitting at or above their fair share. Apart from the addition of a high priority buffer, ECSFQ behaves identically to the original CSFQ algorithm. By providing low queueing delays for flows transmitting at rates lower than their fair share, ECSFQ allows flows that rely on the detection of incipient congestion to increase their transmission rates until a fair share is achieved.

The results of Figure 8 were obtained by repeating the previous experiment shown in Figure 7(b) with ECSFQ and TCP Vegas. Due to the presence of high priority buffers, TCP Vegas packets experience lower queueing delays than the UDP packets, and all three flows achieve approximately fair shares of the bottleneck link bandwidth.

One potential drawback of ECSFQ is that it can introduce discrepancies in packet sequence. We submit, however, that packet reordering will be rare, since it occurs only when a flow's packets are queued in the high priority buffer after

previously being queued in the low priority buffer. Such an event can occur in two cases: (1) when a flow originally transmits at or above its fair share allocation but later decreases its transmission rate below the fair share, or (2) when bandwidth becomes available and the flow's fair share suddenly increases. Packet reordering in the first case is possible but unlikely, because by reducing its rate, the flow is reducing the load on the bottleneck, thereby allowing the packets in the low priority buffer to be processed faster, resulting in a low probability of packets from this flow being found in the low priority buffer. Packet reordering in the second case is also possible but again unlikely, since the low priority buffer empties rapidly when new bandwidth becomes available [4].

5 Simulation Experiments

We now present the results of several simulation experiments, each of which is designed to test a different aspect of Network Border Patrol's performance. The first set of experiments examines the ability of NBP to prevent congestion collapse; the second set of experiments examines the ability of ECSFQ to provide fair bandwidth allocations to competing network flows; and the third set of experiments assesses the scalability constraints of NBP. All simulations were run for 100 seconds using the UC Berkeley/LBNL/VINT ns-2 simulator [31]. The ns-2 code implementing NBP and the scripts to run these simulations are available at the UCI Network Research Group web site [32]. Default simulation parameters are listed in Table 1. They are set to values commonly used in the Internet and are used in all simulation experiments unless otherwise specified.

5.1 Preventing Congestion Collapse

A. Single congested link

The first set of simulation experiments explores NBP's ability to prevent congestion collapse from undelivered packets. Consider the scenario depicted in Figure 9. One flow is a TCP flow generated by an application that always has data to send, and the other flow is a constant bit rate UDP flow generated by an application that is unresponsive to congestion. Both flows compete for access to a shared 1.5 Mbps bottleneck link (R_1-R_2), and only the UDP flow traverses a second bottleneck link (R_2-E_2), which has a limited capacity of 128 kbps.

Figure 10 shows the throughput achieved by the two flows as the UDP source's transmission rate is increased from 32 kbps to 2 Mbps. The combined throughput delivered by the network (i.e., the sum of both flow's throughputs) is also

[4] We have conducted extensive simulations and from our experience, we conclude that packet reordering is theoretically possible, but it practically does not occur and we did not see any impact in performance due to packet reordering. We have simulated the same set of simulations in ns-2 provided by the authors of CSFQ in their original paper [18], using their ns-2 code made available by the authors in their web site, and we have not found any scenario in which ECSFQ provides worse fairness performance than CSFQ.

Simulation parameter	Value
Packet size	1000 bytes
Router queue size	100 packets
Maximum segment size (MSS)	1500 bytes
TCP implementation	Reno [27]
TCP window size	100 kbytes
NBP Quantum factor (QF)	10
NBP T_x	40000 bytes
NBP τ_f	100 msec
TSW window size	10 msec
End-system-to-edge propagation delay	100 μsec
End-system-to-edge link bandwidth	10 Mbps

Table 1: Default simulation parameters

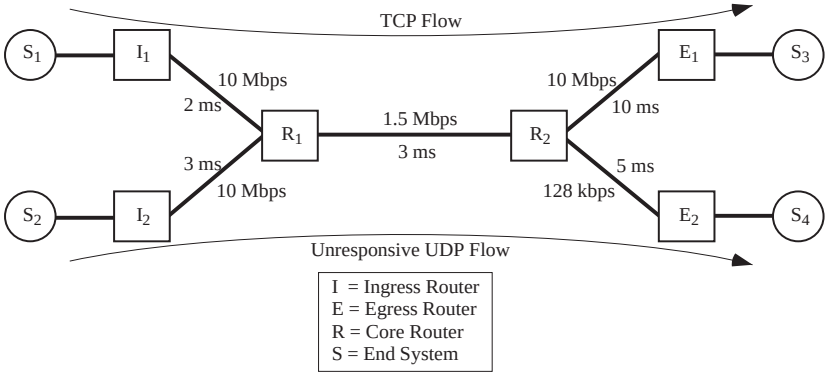
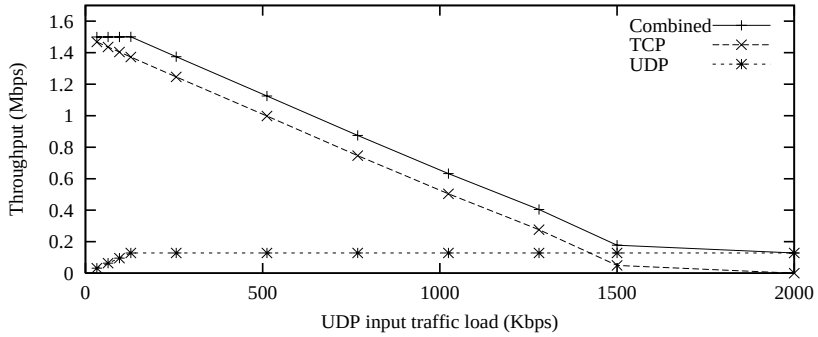


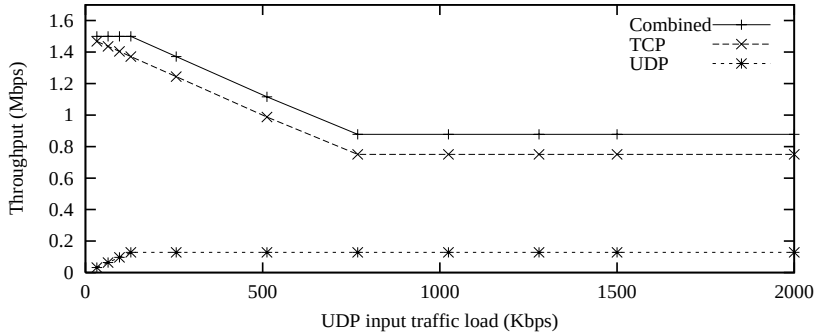
Figure 9: A network with a single shared link

shown. Three different cases are examined under this scenario. The first is the benchmark case used for comparison: NBP is not used between edge routers, and all routers schedule the delivery of packets on a FIFO basis. As Figure 10(a) shows, the network experiences severe congestion collapse as the UDP flow's transmission rate increases, since the UDP flow fails to respond adaptively to the discarding of its packets on the second bottleneck link. When the UDP load increases to 1.5 Mbps, the TCP flow's throughput drops nearly to zero.

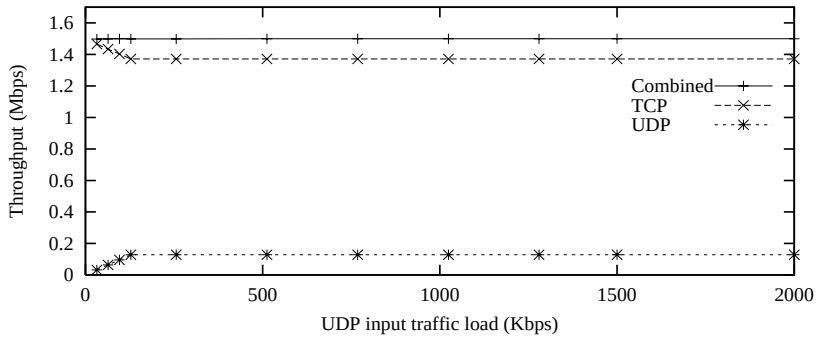
In the second case we show that fair queueing mechanisms alone cannot prevent congestion collapse. As shown in Figure 10(b), better throughput is achieved for the TCP flow when compared to the FIFO-only case. As indicated by the combined throughput of both flows, however, congestion collapse still occurs as the UDP load increases. Although ECSFQ allocates about 750 kbps to each flow at the first bottleneck link, only 128 kbps of this bandwidth is successfully exploited by the UDP flow, which is even more seriously bottlenecked by a second link. The remaining 622 kbps is wasted on undelivered packets. Similar results are observed when ECSFQ is replaced by ordinary fair



(a) Severe congestion collapse using FIFO only



(b) Moderate congestion collapse using ECSFQ only



(c) No congestion collapse using NBP with FIFO

Figure 10: Congestion collapse observed as unresponsive traffic load increases. The solid line shows the combined throughput delivered by the network.

queueing. In the third case, as Figure 10(c) shows, NBP effectively eliminates congestion collapse: the TCP flow achieves a nearly optimal throughput of 1.37 Mbps, and the combined throughput remains very close to 1.5 Mbps.

B. Traversing multiple congested links

In this experiment, we examine whether NBP effectively prevents congestion collapse when a TCP flow traverses several bottleneck links that also support traffic from unresponsive UDP flows. The network configuration used for

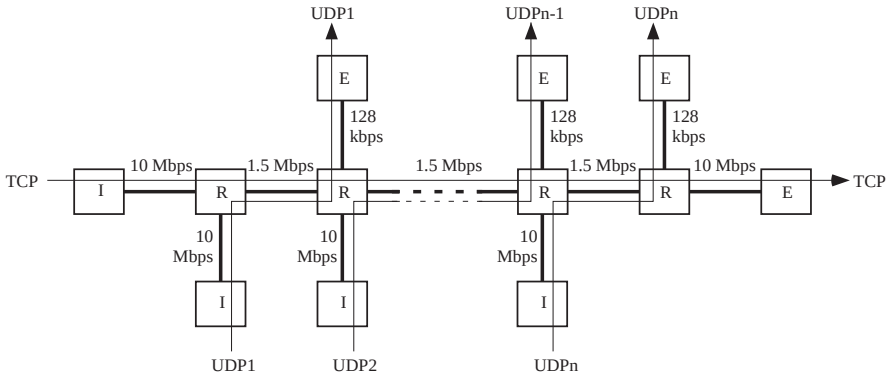


Figure 11: A network with multiple congested router hops

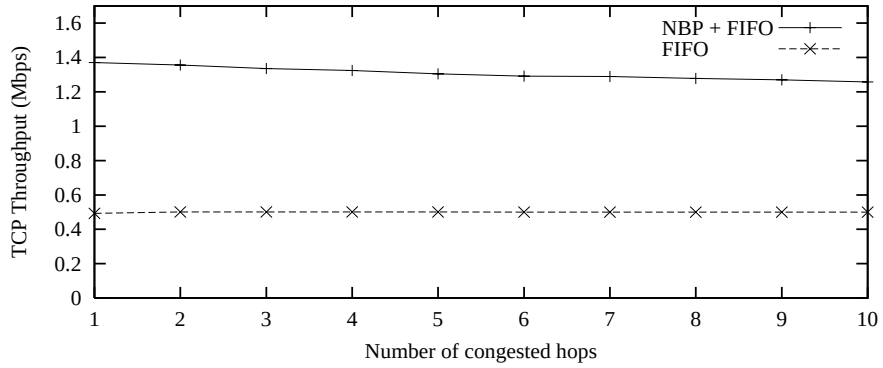


Figure 12: TCP throughput in a network with multiple congested router hops

this simulation experiment is shown in Figure 11. In this configuration, a TCP flow shares several 1.5 Mbps bottleneck links with unresponsive UDP flows. Each of these bottleneck links is further bottlenecked by another link with a capacity of 128 kbps. All links have propagation delays of 10 msec, and each UDP source transmits packets at a constant rate of 1 Mbps.

Figure 12 shows the throughput of the TCP flow as the number of congested router hops increases from 1 to 10. In the ideal scenario, the transmission rate of each UDP source is regulated by an ingress router to 128 kbps, and the throughput for the TCP flow becomes 1.37 Mbps. When only FIFO scheduling is used, UDP sources consume 1 Mbps of the bottleneck links' bandwidth, even though UDP sources are bottlenecked downstream at the 128 kbps links, and therefore, the TCP flow is limited to a throughput of approximately 0.5 Mbps regardless of the number of hops, whereas NBP allows the network to avoid congestion collapse, allocating nearly 1.37 Mbps to the TCP flow when the number of hops is small. As the number of hops increases, the throughput of the TCP flow diminishes slightly due to increased feedback delays between the the ingress and egress routers supporting the TCP flow.

5.2 Achieving Fairness

Network Border Patrol, on its own, does not achieve fair bandwidth allocations to competing network flows. When combined with ECSFQ, however, we hypothesize that it does. To test this hypothesis, we perform two fairness experiments. In the first fairness experiment, we consider the scenario depicted in Figure 9, with the second bottleneck link (R_2-E_2) replaced by a higher capacity 10 Mbps link. This leaves the network with only a single bottleneck link (R_1-R_2). The TCP flow is generated by an application that always has data to send, and the UDP flow is generated by an unresponsive source which transmits packets at a constant bit rate.

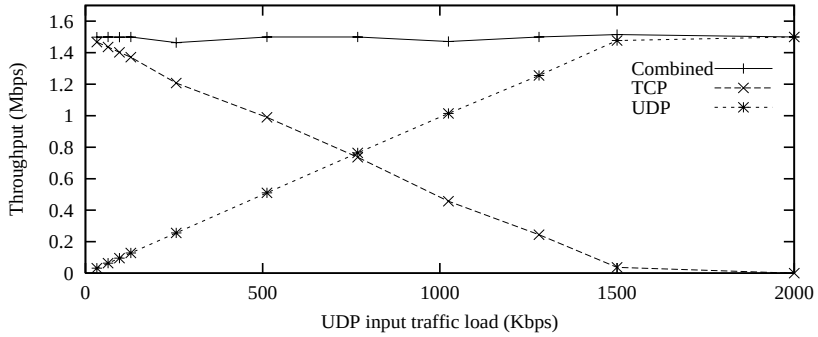
Since there is only one bottleneck link in this scenario, the max-min fair allocation of bandwidth between the flows is 750 kbps (as long as the UDP source exceeds a transmission rate of 750 kbps). However, as Figure 13(a) shows, fairness is clearly not achieved when only FIFO scheduling is used in routers. (NBP is not used in this simulation scenario.) As the unresponsive UDP traffic load increases, the TCP flow experiences congestion and reduces its transmission rate, thereby granting an unfairly large amount of bandwidth to the unresponsive UDP flow. Thus, although there is no congestion collapse from undelivered packets, as indicated by the constant combined network throughput, there is clearly unfairness between the TCP and UDP flows.

When NBP is deployed with FIFO scheduling, Figure 13(b) shows that the unfair allocation of bandwidth is only slightly reduced, since NBP has no explicit mechanism to provide fairness. Figure 13(c) shows the throughput of each flow when ECSFQ is used (without NBP). Notice that ECSFQ is able to approximate fair bandwidth allocations.

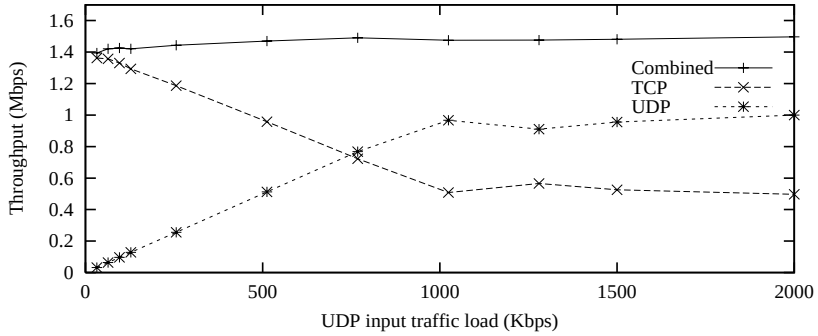
In the second fairness experiment, we study whether NBP combined with ECSFQ provides max-min fairness in a complex network. We consider the network model shown in Figure 14. This model is adapted from the second General Fairness Configuration (GFC-2), which is specifically designed to test the max-min fairness of traffic control algorithms [33]. It consists of 22 unresponsive UDP flows, each generated by a source transmitting at a constant bit rate of 100 Mbps.

Flows belong to flow groups, which are labeled from A to H, and the network is designed in such a way that, if max-min fairness is achieved, members of each flow group receive the same max-min bandwidth allocations. Links connecting core routers serve as bottlenecks for at least one of the 22 flows, and all links have propagation delays of 5 msec and bandwidths of 150 Mbps unless otherwise shown in the figure.

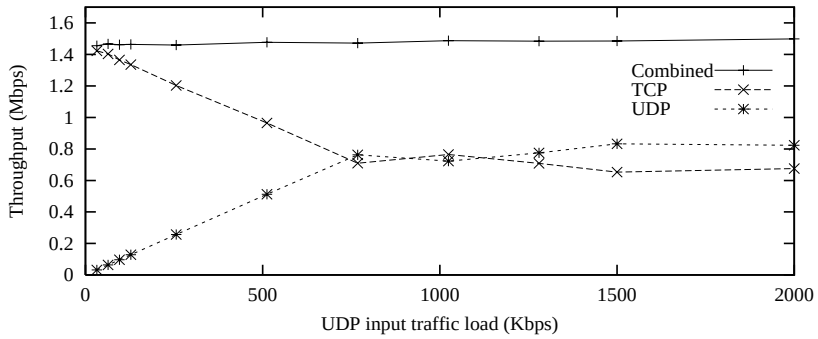
The second column of Table 2 lists the ideal (mathematically calculated) global max-min fair share allocations for each flow group shown in Figure 14. These values represent the ideal bandwidth allocations for any traffic control mechanism that attempts to provide global max-min fairness. The remaining columns list the throughputs observed after 4.5 seconds of simulation for several scenarios. (The average results for each flow group are shown.) In the



(a) Severe unfairness using FIFO only



(b) Moderate unfairness using NBP with FIFO



(c) Approximate fairness using ECSFQ

Figure 13: Unfairness as the unresponsive traffic load increases

first simulation scenario, NBP is not used, and all routers implement a standard (complex) core-stateful fair queuing algorithm, namely weighed fair queueing (WFQ).

As indicated by the throughput values in the second column (i.e., ideal case) and third column (i.e., fair queueing only case), weighed fair queuing by itself is unable to achieve global max-min fairness for all flow groups. This is due to the fact that fair queuing cannot by itself prevent congestion collapse. In the second simulation scenario, NBP is introduced at edge routers and FIFO scheduling is assumed at all routers. Results for this simulation scenario are listed in the third column of Table 2 and show that NBP with FIFO also fails to achieve global max-min fairness in the

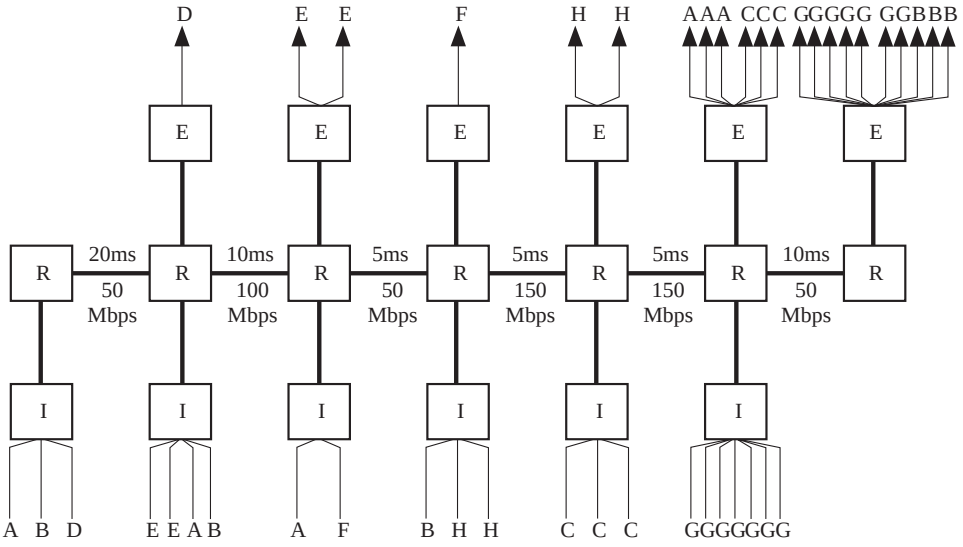


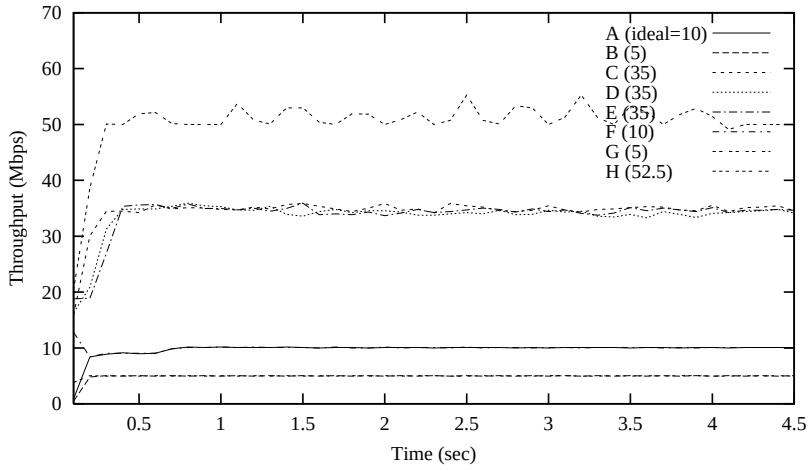
Figure 14: The GFC-2 network

Flow Group	Ideal global max-min fair share (Mbps)	Simulation results							
		Throughput using FQ only		Throughput using NBP with FIFO		Throughput using NBP with WFQ		Throughput using NBP with ECSFQ	
		(Mbps)	Normalized	(Mbps)	Normalized	(Mbps)	Normalized	(Mbps)	Normalized
A	10	8.32	0.83	10.96	1.09	10.00	1.00	10.40	1.04
B	5	5.04	1.01	1.84	0.36	5.04	1.01	4.48	0.90
C	35	27.12	0.77	31.28	0.89	34.23	0.98	31.52	0.90
D	35	16.64	0.47	33.84	0.96	34.95	0.99	32.88	0.94
E	35	16.64	0.47	37.76	1.08	34.87	0.99	33.36	0.95
F	10	8.32	0.83	7.60	0.76	10.08	1.00	8.08	0.80
G	5	4.96	0.99	1.04	0.20	4.96	0.99	5.28	1.05
H	52.5	36.15	0.69	46.87	0.90	50.47	0.97	47.76	0.91

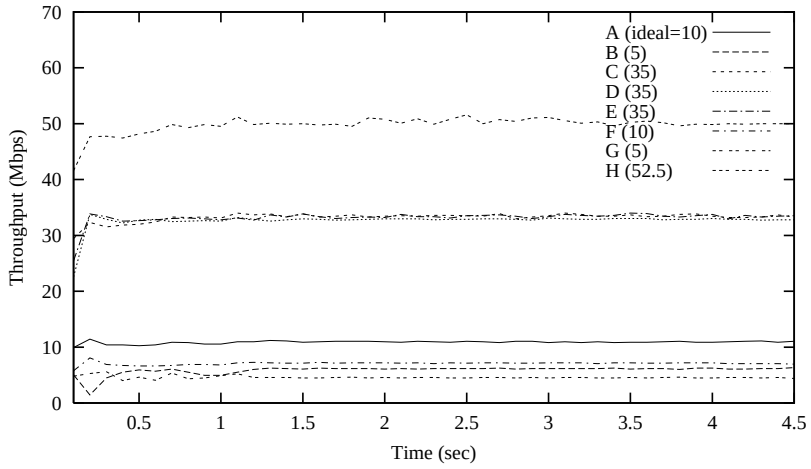
Table 2: Per-flow throughput in the GFC-2 network

GFC-2 network. This is largely because NBP, by itself, has no mechanism to explicitly enforce fairness.

In the third and fourth simulation scenarios, NBP is combined with the standard fair queuing, namely weighed fair queueing (WFQ), and ECSFQ, respectively. Simulation results show that in both cases bandwidth allocations are approximately max-min fair for all flow groups. NBP with standard fair queuing achieves slightly better fairness than NBP with ECSFQ, since ECSFQ is only an approximation of fair queuing and its performance depends on the accuracy of its estimation of a flow's input rate and fair share. Remember that NBP with ECSFQ achieves approximate global max-min fairness and ECSFQ is much less complex than WFQ, due to it is stateless design. Figures 15(a) and 15(b) depict how rapidly the throughput of each flow converges to its max-min fair bandwidth allocation for the NBP



(a) Using NBP with WFQ



(b) Using NBP with ECSFQ

Figure 15: Per-flow throughput in the GFC-2 network

with fair queuing and the NBP with ECSFQ cases, respectively. Even in a complex network like the one simulated here, all flows converge to an approximately max-min fair bandwidth allocation within one second.

5.3 Scalability

Scalability is perhaps the most important performance measure of any traffic control mechanism. As we have just seen, NBP is a core-stateless traffic control mechanism that effectively prevents congestion collapse and provides approximate max-min fairness when combined with an appropriate fair queuing mechanism. However, NBP's scalability is highly dependent upon per-flow management performed by edge routers. In a large scale network, the overheads of maintaining per-flow state, communicating per-flow feedback, and performing per-flow rate control and rate monitoring may become inordinately expensive. The load of feedback packets generated by an NBP ingress router depends on

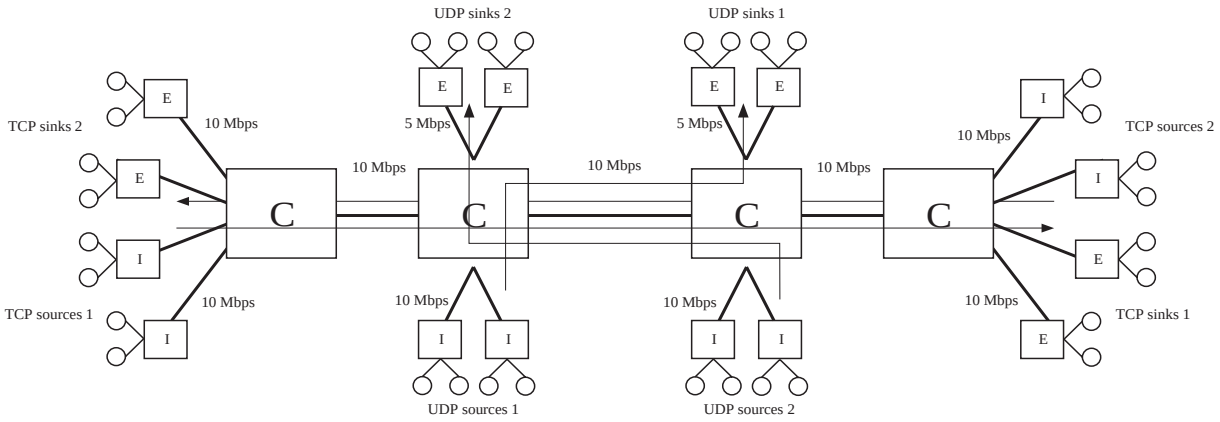


Figure 16: Simulation model for evaluating scalability

the number of egress routers it communicates with and the load of the ingress traffic. The processing required at border routers and the amount of state that border routers need to maintain depend on the number of active flows. Hence, the scalability of NBP is dependent on the number of border routers, the number of active flows and the traffic load.

A. Multiple flows and multiple border routers

In the set of experiments shown in this subsection, we assess NBP's scalability using the network shown in Figure 16. The number of border routers and the number of active flows per border router is varied in our simualtions. The network model consists of four core routers, $4 \times B$ border routers, and $4 \times B \times F$ flows, where B is the number of border routers per core router and F is the number of flows per border router. Propagation delays are 5 msec between core routers, 1 msec between border and core routers, and 100 μsec between end systems and core routers. Flows are established in both directions so that data packets travel in both directions on all links connecting core routers. TCP flows traverse all core routers while UDP flows traverse only the interior core routers. The capacities of links between core and egress routers traversed by UDP flows are set to 5 Mbps, while all remaining link capacities are set to 10 Mbps. Thus, UDP flows are bottlenecked at 5 Mbps. TCP flows traverse multiple congested links, and compete for bandwidth with UDP flows and also among themselves. UDP flows are unresponsive to congestion and transmit at a constant rate of 5 Mbps.

In the first experiment we consider a moderately large network with 8 border routers and vary the number of flows from 8 to 48. The amount of feedback generated by NBP is shown in Figure 17. This figure shows that the amount of feedback is mostly independent of the number of flows, and thus, NBP is scalable in this scenario. This is due to the fact that, in NBP, feedback packets are generated according to the number of packets admitted into the network, not according to the number of flows admitted into the network. Regardless of the number of flows, since in nearly all test

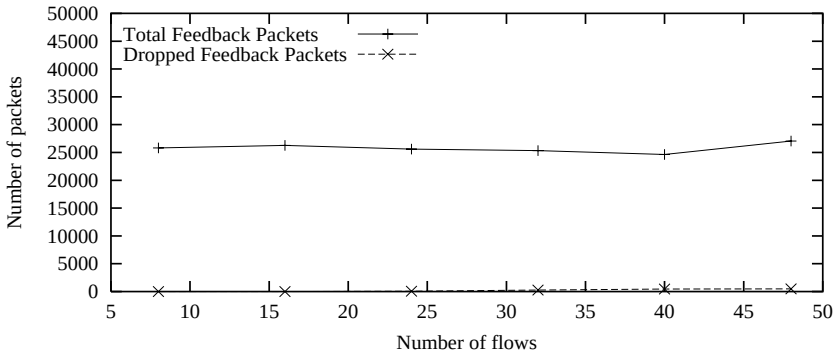
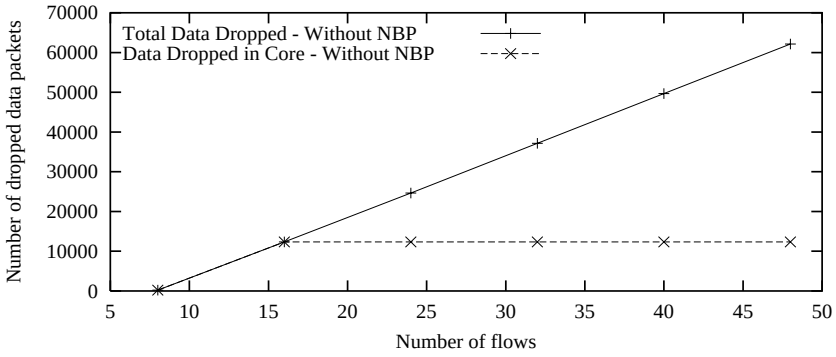
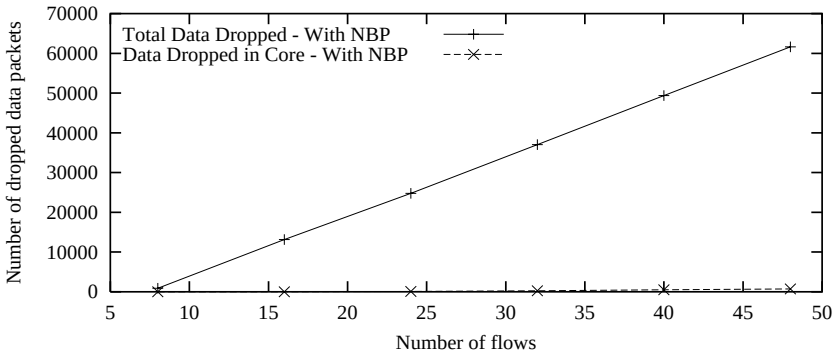


Figure 17: Feedback traffic overhead as the number of flows increases.



(a) Using FIFO only



(b) Using NBP with FIFO

Figure 18: Number of dropped packets as the number of flows increases.

simulation scenarios the capacity of the network is fully utilized, the number of packets admitted into the network is practically constant and the amount of feedback information generated by NBP does not increase with the number of flows. Furthermore, simulation results show that the fraction of the bandwidth of the links in the core of the network consumed by feedback packets is reasonably small and varies from 1.04% to 1.59% in this experiment.

NBP prevents congestion collapse by eliminating or severely reducing the number of packet losses within the core of the network. Figure 18 shows the total number of data packets lost in the network with and without NBP. As expected,

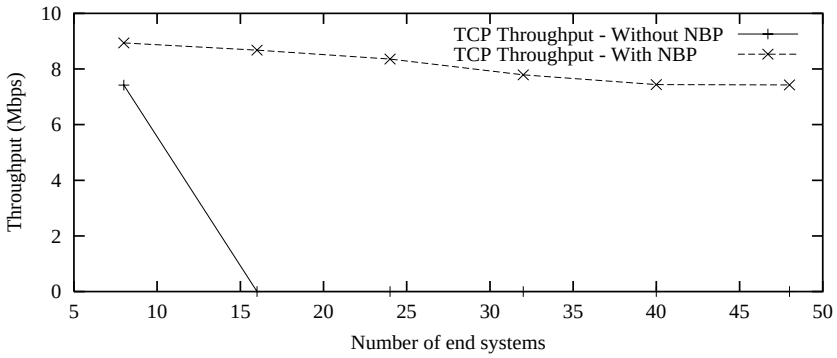


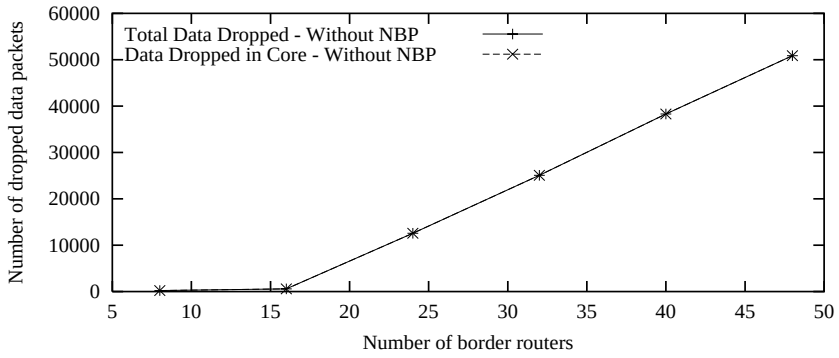
Figure 19: Combined throughput of TCP flows as the number of flows increase.

the total number of packet losses increases linearly with the number of flows according to the load of unresponsive traffic. However, without NBP, a significant portion of these losses occur at core routers, whereas with NBP, nearly all packet losses occur at the ingress border of the network. Since packet losses in the core of the network often lead to congestion collapse due to undelivered packets, NBP (i.e., dropping packets at the entry point to the network) is clearly advantageous. Moreover, NBP is able to limit packet losses to the borders of the network even as the number of active flows increases. Therefore,

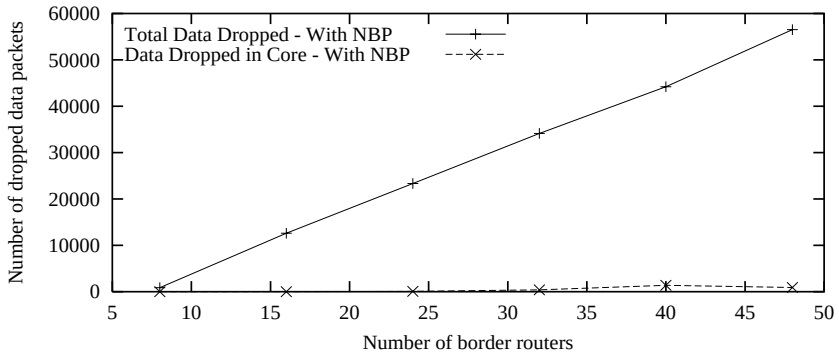
Figure 18 shows that NBP effectively prevents congestion collapse in a scalable manner when the number of flows increases.

In the simulation configuration shown in Figure 16, TCP flows traversing the entire network compete for bandwidth with the unresponsive UDP flows. The optimal throughput of TCP flows is 5 Mbps in the forward path and 5 Mbps in the reverse path. The optimal combined throughput of TCP flows is, thus, 10 Mbps in this network. Figure 19 shows that without NBP, the combined throughput of TCP flows drops to nearly zero as the UDP unresponsive traffic load increases. Figure 19 also illustrates that, with NBP, TCP's throughput remains close to optimal even as the number of flows increases. This is because, as seen in Figure 18, NBP is able to prevent congestion collapse, and therefore, the performance of TCP is greatly improved.

In the second experiment, we vary the size of the network by varying the number of border routers from 8 to 48. We attach only one end system to each border router so that links between ingress and core routers are never congested; only links connecting core routers and links connecting core to egress routers may become congested. Figures 20(a) and (b) show the number of data packets lost in the network with and without NBP. As in the first experiment, the total number of packet losses increases linearly with the number border routers according to the load of unresponsive traffic. Without NBP, all packet losses occur in the core of the network, whereas with NBP, nearly no losses are observed in the core.



(a) Using FIFO only



(b) Using NBP with FIFO

Figure 20: Number of dropped packets as the number of border routers increase.

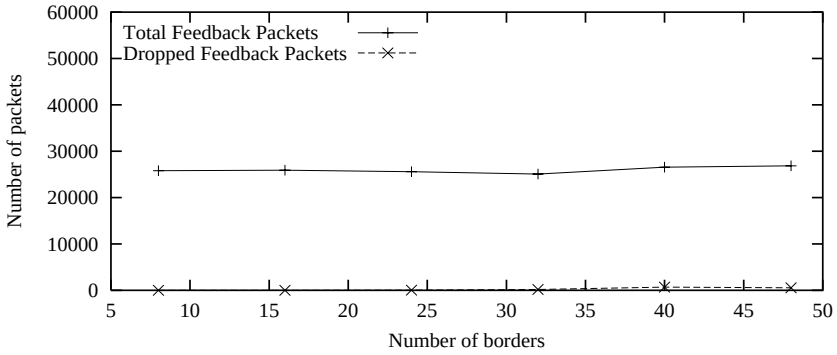


Figure 21: Amount of feedback overhead as the number of border routers increases.

For the network used in Figure 16, the amount of feedback information generated by NBP is shown in Figure 21. This Figure shows that the feedback packet overhead that NBP creates remains independent of the number of border routers. Simulation result also shows that the fraction of the bandwidth of links L2 and L3 consumed by feedback packets is reasonably small and varies from 0.92% to 1.41% in this experiment. The fact that the feedback overhead that NBP generates is relatively small and remains independent of the number of border routers (i.e., the size of the network) suggest that NBP scales well.

Note that even in large scale networks, nearly no packet is lost at the core of the network. This demonstrates NBP effectively prevents congestion collapse by dropping packets before they enter the network.

B. Ingress routers communicating with multiple egress routers

Ingress routers may exchange feedback packets with a large number of egress routers, if flows originating at an ingress router are destined to various egress routers in the network. In this experiment, the network model of Figure 22 is used to evalute NBPs scalability with respect to the number of destination egress routers. Propagation delays are 10 msec between core routers, 1 msec between border and core routers, and 100 μsec between end systems and border routers. The capacities of links between core routers are set to 20 Mbps, between core routers and egress routers connected to UDP sinks are set to 5 Mbps, while all remaining link capacities are set to 10 Mbps. TCP flows and unresponsive UDP flows traverse the entire network and compete for bandwidth among themselves. Flows are configured such that, ingress routers on the left of the network exchange feedback packets with all egress routers on the right. Conversely, ingress routers on the right of the network need to exchange feedback packets with all egress routers on the left. In this simulation scenario, the optimal TCP throughput is 10 Mbps and the maximum UDP throughput is 5 Mbps.

In this simulation experiment, the load of UDP traffic is varied from 2 Mbps to 20 Mbps and Table 3 shows the simulation results using NBP. As shown in the third column (UDP throughput), as the UDP traffic load increases, NBP successfully limits the UDP throughput to 5 Mbps. TCP throughput is shown in the second column and remains close to optimal for all UDP loads. The fourth column shows that the number of feedback packets increases up to the point where the network becomes fully utilized. As the load of UDP traffic increases beyond 6 Mbps, the number of feedback packets remain approximately constant. The remaining columns show how many packets are dropped and where they are dropped. As the last column shows no data packet is dropped in the core.

As the UDP traffic load increases, more data packets are dropped by ingress routers, in order to prevent congestion collapse within the network.

The simulation results show that the NBP, even when ingress routers exchange feedback information with multiple egress routers, the number of feedback packets is bounded and NBP remains effective in preventing congestion collapse, and thus, in this scenario, NBP scales very well.

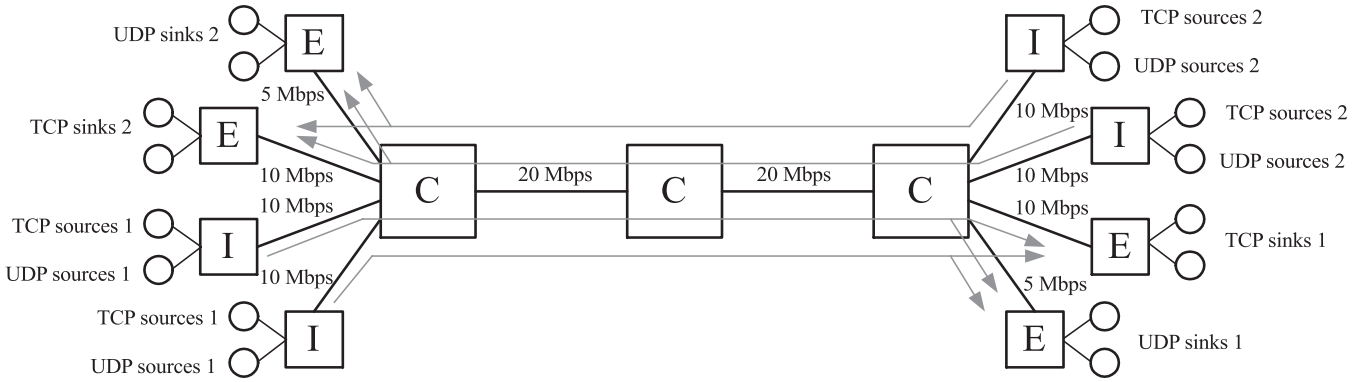


Figure 22: Simulation model for evaluating ingress routers communicating with multiple egress routers

UDP load (Mbps)	TCP throughput (Mbps)	UDP throughput (Mbps)	Number of feedback packets	Number of dropped packets				
				Data packets	Feedback packets	Total packets	TCP ACK packets	In the Core
2	9.99	2.00	1642	0	25	25	0	0
4	9.98	3.99	2071	0	171	171	0	0
6	9.98	4.92	2234	0	1230	1230	0	0
8	9.98	4.98	2260	0	3660	3660	0	0
10	9.98	4.99	2303	0	6163	6163	0	0
12	9.97	4.99	2332	0	8780	8780	0	0
16	9.97	4.99	2327	0	13756	13756	0	0
20	9.94	4.99	2315	0	18615	18615	0	0

Table 3: Performance results for ingress routers communicating with multiple egress routers

6 Implementation Issues

A number of important implementation issues must be addressed before NBP can be feasibly deployed in the Internet. Some of such issues are discussed below:

1. *Scalable flow classification.* To reduce the overhead of maintaining state for individual flows at edge routers, it may be useful in some cases to aggregate flows through coarser forms of flow classification. Instead of classifying a flow using the packet's addresses and port numbers, the network's edge routers may aggregate many flows together by, for instance, classifying them using only the packet's address fields. Alternatively, flows may be even more coarsely classified using only the packet's destination network address. Coarse-grained flow aggregation has the effect of significantly reducing the number of flows seen by NBP edge routers, thereby reducing the required amount of state and processing required at edge routers. A potential drawback of flow aggregation, however, is that adaptive flows aggregated with unresponsive flows may be indiscriminately punished by an ingress router. The trade-offs between coarse and fine grained flow classification must be assessed by each

network operator.

2. *Scalable inter-domain deployment.* An approach to further improving the scalability of NBP, inspired by a suggestion in [18], is to develop trust relationships between domains that deploy NBP. The inter-domain router connecting two or more mutually trusting domains may become a simple NBP core router without the need to perform per-flow tasks or maintain per-flow state. If a trust relationship cannot be established, border routers between the two domains may exchange congestion information so that congestion collapse can be prevented not only within a domain, but throughout multiple domains.

3. *Incremental deployment.* It is crucial that NBP be implemented in all edge routers of an NBP-capable network. If one ingress router fails to police arriving traffic or if one egress router fails to monitor departing traffic, NBP will not operate correctly, and congestion collapse will be possible. Nevertheless, it is not necessary for *all* networks in the Internet to deploy NBP in order for it to be effective. Any network that deploys NBP will enjoy the benefits of eliminated congestion collapse within the network. Hence, it is possible to incrementally deploy NBP into the Internet on a network-by-network basis.

4. *Multicast.* Multicast routing makes it possible for copies of a flow's packets to leave the network through more than one egress router. When this occurs, an NBP ingress router must examine backward feedback packets returning from each of the multicast flow's egress routers. To determine whether the multicast flow is experiencing congestion, the ingress router should execute its rate control algorithm using backward feedback packets from the most congested ingress-to-egress path (i.e., the one with the lowest flow egress rate). This has the effect of limiting the ingress rate of a multicast flow according to the most congested link in the flow's multicast tree.

5. *Multi-path routing.* Multi-path routing makes it possible for packets from a single flow to leave the network through different egress routers. In order to support this possibility, an NBP ingress router may need to examine backward feedback packets from more than one egress router in order to determine the combined egress rate for a single flow. For a flow passing through more than one egress router, its combined egress rate is equal to the sum of the flow's egress rates reported in backward feedback packets from each egress router.

6. *Integrated or differentiated services.* NBP treats all flows identically, but integrated and differentiated services networks allow flows to receive different qualities of service. In such networks, NBP should be used to regulate best effort flows only. Flows using network services other than best effort are likely to be policed by other more service-appropriate traffic control mechanisms.

7 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. ECSFQ complements NBP by providing fair bandwidth allocations in a core-stateless fashion. NBP ensures at the border of the network that each flow's packets do not enter the network faster than they are able to leave it, while ECSFQ ensures, at the core of the network that flows transmitting at a rate lower than their fair share experience no congestion, i.e., low network queueing delay. This allows the transmission rate of all flows to converge to the network fair share.

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, while ECSFQ requires a simple core-stateless modification to core routers.

Simulation results 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.

[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.

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

[5] "Real Video 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] D. Hong, C. Albuquerque, C. Oliveira, and T. Suda, "Evaluating the Impact of Emerging Streaming Media Applications on TCP/IP Performance," *IEEE Communications Magazine*, vol. 39, no. 4, pp. 76–82, Apr. 2001.

[8] J. Wu, M. Hassan, and S. Jha, "QPD: A Packet Discard Mechanism for Multimedia Traffic in the Internet," in *IEICE Transactions on Communications*, 2001, pp. 538–541.

[9] A. Habib and B. Bhargava, "Unresponsive Flow Detection and Control in Differentiated Services Networks," in *13th IASTED International Conference on Parallel and Distributed Computing and Systems*, Aug. 2001.

[10] A. Mustafa and M. Hassan, "End to End IP Rate Control," in *Recent Advances in Computing and Communications*. Dec. 2000, pp. 279–282, Tata McGraw-Hill Publishing Company Limited.

[11] A. Rangarajan and A. Acharya, "ERUF: Early Regulation of Unresponsive Best-Effort Traffic," *International Conference on Networks and Protocols*, October 1999.

[12] M. Parris and K. Jeffay, "A Better-Than-Best-Effort Service for Continuous Media UDP Flows," in *NOSSDAV*, 1998.

[13] S. Robinson, "Multimedia Transmission Drive Net Toward Gridlock," in *New York Times*, August 23 1999.

[14] 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.

[15] 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.

[16] B. Suter, T.V. Lakshman, D. Stiliadis, and A. Choudhury, "Design Considerations for Supporting TCP with Per-Flow Queueing," in *Proc. of IEEE Infocom*, March 1998, pp. 299–305.

[17] B. Braden *et al.*, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, IETF, April 1998.

[18] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proc. of ACM SIGCOMM*, September 1998, pp. 118–130.

[19] Z. Cao, Z. Wang, and E. Zegura, "Rainbow Fair Queuing: Fair Bandwidth Sharing Without Per-Flow State," in *Proc. of IEEE Infocom*, March 2000.

[20] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe - A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," in *Proc. of IEEE Infocom*, March 2000.

[21] D. Lin and R. Morris, "Dynamics of Random Early Detection," in *Proc. of ACM SIGCOMM*, September 1997, pp. 127–137.

[22] D. Bertsekas and R. Gallager, *Data Networks, second edition*, Prentice Hall, 1987.

[23] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and R. Viswanathan, "ERICA Switch Algorithm: A Complete Description," ATM Forum Document 96-1172, Traffic Management WG, August 1996.

[24] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," Request for Comments 2475, Internet Engineering Task Force, December 1998.

[25] D. Clark and W. Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362–373, August 1998.

[26] K.K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *ACM Transactions on Computing Systems*, vol. 8, no. 2, pp. 158–181, May 1990.

[27] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, IETF, January 1997.

[28] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. of ACM SIGCOMM*. ACM, Aug. 1994, pp. 34–35.

[29] U. Hengartner, J. Bolliger, and Th. Gross, "TCP Vegas Revisited," in *Proc. of IEEE Infocom*, March 2000.

[30] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communications Review*, vol. 24, no. 5, pp. 8–23, Oct. 1994.

[31] LBNL Network Research Group, *UCB/LBNL/VINT Network Simulator - ns (version 2)*, http://www-mash.cs.berkeley.edu/ns/, September 1997.

[32] UCI Network Research Group, *Network Border Patrol (NBP)*, http://netresearch.ics.uci.edu/nbp/, 1999.

[33] B. Vandalore, S. Fahmy, R. Jain, R. Goyal, and M. Goyal, "A Definition of Generalized Fairness and its Support in Switch Algorithms," ATM Forum Document 98-0151, Traffic Management WG, February 1998.

[34] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit Window Adaptation: A Method to Enhance TCP Performance," in *Proc. of IEEE Infocom*, April 1998.

[35] C. Albuquerque, B.J. Vickers, and T. Suda, "Network Border Patrol," in *Proc. of IEEE Infocom*, March 2000.

[36] W.K. Tsai and Y. Kim, "Re-Examining Maxmin Protocols: A Fundamental Study on Convergence, Complexity, Variations, and Performance," in *Proc. of IEEE Infocom*, April 1999, pp. 811–818.

Appendix: NBP's Feedback Overhead

Feedback packets assist NBP to prevent congestion collapse by carrying congestion control information between ingress and egress routers. However, feedback packets themselves increase the traffic load and may contribute to congestion within the network. In the following, we present rough estimate of the amount of overhead that NBP feedback packets create.

In order to estimate the load of feedback packets within the network, we estimate how often feedback packets are generated. The frequency that an ingress router i generates a feedback packet destined to egress router e is:

$$\lambda_{ie} = \frac{f_{ie}.ingressRate}{T_x} \tag{1}$$

where T_x is NBP's byte transmission counter threshold, i.e., the number of bytes of flow f_{ie} between feedback packets; and $f_{ie}.ingressRate$ is the allowed input rate for flows between ingress router i and egress router e.

Feedback packets may be generated between any pair of ingress and egress routers, and in the worst case, ingress routers exchange feedback packets with all egress routers in the network. In this worst case, the total rate at which feedback packets are generated in a network is:

$$\Lambda = \sum_{i=0}^{I} \sum_{e=0}^{E} \lambda_{ie} \tag{2}$$

or

$$\Lambda = \frac{1}{T_x} \sum_{i=0}^{I} \sum_{e=0}^{E} f_{ie}.ingressRate \tag{3}$$

where I and E are the total number of ingress and egress routers in the network, respectively.

Consider the bottleneck link bandwidth between all ingress routers and all egress routers to be B. Since NBP's rate control algorithm is capable of preventing congestion in the network, the sum of all ingress rates is smaller than or equal to B. Thus, the following inequality holds.

$$\sum_{i=0}^{I} \sum_{e=0}^{E} f_{ie}.ingressRate < B \tag{4}$$

Applying the above equation to eq.(3), the maximum rate at which feedback packets are generated in a network is given by:

$$\Lambda = \frac{B}{T_x} \tag{5}$$

Assuming that the average length of feedback packets is L, the total load created by the feedback packets in the network is:

$$\Gamma = L \times \Lambda \tag{6}$$

Feedback packets in NBP take the form of ICMP packets, and in the case where only one flow information is maintained between a pair of ingress-egress routers, forward feedback packets are 28 bytes long, and backward feedback packets are 36 bytes long [5]. Therefore, for a network with a bottleneck bandwidth B of 10 Mbps, feedback packets of length 28 bytes, and NBP T_x parameter set to 10,000 bytes, the total load of feedback packets in the network is Γ equals to 28 kbps.

[5] In the general case, feedback packets may have variable length, depending on the number of flows between a pair of ingress and egress routers. The most scalable scenario is when all packets flowing between a given pair of ingress and egress routers are classified into the same flow, and thus, a feedback packet contains information on only one flow.