Adaptive Congestion Detection and Control at the Application Level for VoIP

Teck-Kuen Chua and David C. Pheanis
Computer Science and Engineering
Ira A. Fulton School of Engineering
Arizona State University
Tempe, Arizona 85287-8809
Email: TeckKuen.Chua@gmail.com or David.Pheanis@ASU.edu

Abstract

For decades, researchers have worked extensively in the area of congestion control for packet-switched networks. Many proposed solutions take advantage of the congestion-control mechanism in Transmission Control Protocol *(TCP), and these approaches work well for networks that have heavy TCP traffic. However, these approaches are not universally effective — they fail completely for protocols that do not implement congestion-control mechanisms. In particular, these approaches do not work with the* User Datagram Protocol *(UDP). Real-time media-streaming technologies such as* Voice over Internet Protocol *(VoIP) and video conferencing use UDP and therefore do not respond well to existing congestion-avoidance techniques. We propose a new, adaptive, responsive, end-to-end technique to implement application-level congestion detection and control for real-time applications such as VoIP. Unlike existing methods, which rely on packet loss as a signal to reduce the transmission rate, our solution proactively reacts to network congestion to* prevent *packet loss, thus improving the QoS of applications that employ our algorithm.*

Keywords: VoIP, QoS, congestion detection, congestion avoidance, adaptive transmission control, real-time media.

1. Introduction

In a packet-switched network, a router connects multiple ingress streams to various egress ports. When a heavy burst of network traffic occurs, congestion builds up at the routers, which form the bottlenecks of the network. When congestion in a router becomes severe enough, the incoming packets consume all of the buffer resources in the router and leave no room for additional inbound packets, thus resulting in lost packets.

The output rate at an egress queue in a router can be only as fast as the serialization rate of the hardware. When an egress queue receives packets from multiple ingress ports at a combined rate that is higher than the serialization rate of the egress port, the egress queue grows, eventually causing congestion. When the egress queue becomes full, the router has to discard all additional packets destined for that egress queue. This dropping of packets is an unwelcome condition known as *tail drop*.

Dropped packets are undesirable at any time, of course, but tail drop is especially undesirable because it leads to an adverse effect called *global synchronization*. Global synchronization occurs when tail drop causes all of the transmitting devices to receive congestion signals at the same time. The transmitting devices consequently reduce their transmission rates in unison, and the link utilization quickly falls well below the optimal level due to the sudden, simultaneous reduction in transmission rates from all of the senders. When congestion eases, the transmitting devices increase their transmission rates all at once, leading to another episode of severe congestion in the network.

Researchers have done considerable work in this area and have proposed numerous solutions to manage and avoid congestion in a packet-switched network. Proposed congestion-avoidance methods include *Random Early Detection* (RED) [1] and its variants [2] [3] [4] [5], and proposed approaches also include BLUE [6], *Stochastic Fair BLUE* (SFB) [7], *Generalized Random Early Evasion Network* (GREEN) [8], and *Explicit Congestion Notification* (ECN) [9]. Each of these suggested techniques exploits the transmission-control mechanism in *Transmission Control Protocol* (TCP). Since real-time IP applications generally use the *User Datagram Protocol* (UDP) as the transport protocol, however, these proposed congestion-control approaches are almost entirely ineffective at curbing real-time media traffic.

We propose a new application-level adaptive congestion-detection and congestion-control mechanism for avoiding congestion with real-time IP applications such as *Voice over Internet Protocol* (VoIP)[1]. We start by explaining how existing congestion-avoidance algorithms work with TCP, and we discuss the problems that existing approaches have with UDP and the intrinsic challenges of congestion control with real-time applications. Then we present our new application-level adaptive congestion-detection and congestion-control solution for real-time IP applications, and we provide measurements that demonstrate the effectiveness of our system.

2. Existing approaches

Congestion has been a serious problem in packet-switched networks since packet switching first came into existence. The importance of the congestion problem is evident from the extensive amount of work that researchers have done to provide ways of avoiding or at least managing congestion. In this section we examine a few well-known congestion-avoidance techniques and show how these methods alleviate congestion in packet-switched networks, so this section provides a basis for understanding the new real-time congestion-control technique that we present later.

2.1. Random Early Detection (RED)

Random Early Detection (RED) is an active queue-management (AQM) technique with the aim of achieving low average delay and high throughput [1]. RED uses the average queue size as an early indication of congestion and signals the transmitting devices to reduce their transmission rates temporarily before the congestion actually occurs. When the average queue size exceeds a predetermined minimum threshold, RED starts dropping randomly selected packets. The probability of dropping packets increases either linearly or exponentially as the average queue size grows beyond the minimum threshold. When the average queue size passes a predetermined maximum threshold, RED starts dropping *all* incoming packets. RED uses the average queue size over some period of time instead of using the instantaneous queue size, so the technique can absorb spikes or short bursts of network traffic without overreacting.

When a TCP host detects packet loss, the host temporarily slows down its transmission rate. TCP increases its transmission rate quickly when all packets reach the destination, an indication that the period of congestion has passed. RED randomly selects packets to drop, thereby distributing the packet loss among assorted hosts at various times. As a result, the hosts reduce their transmission rates at different times and avoid global synchronization.

Weighted RED (WRED) incorporates the IP precedence feature into the RED algorithm. WRED gives preferential handling to packets that have higher priority. When congestion is building, WRED randomly selects packets with lower priority as the first packets to discard. This scheme creates differentiated QoS characteristics for different classes of service.

[1]Patent pending

Some researchers have proposed dynamic RED implementations that adapt to ever-changing network conditions. For example, *adaptive RED* [2], *Dynamic Weighted Random Early Drop* (DWRED) [3], and other similar approaches are adaptive RED variants that are designed to address the sensitivity weaknesses of RED. Both the throughput and the average queue size of RED are very sensitive to the traffic load and RED parameters [10][11], so RED produces unpredictable results in volatile network environments.

Flow-based RED (FRED) [4] includes a mechanism to enforce fairness in resource utilization for each active flow of traffic. This technique uses information such as destination addresses, source addresses, and ports to classify traffic into different flows. The flow-based algorithm maintains state information for every active flow, and the algorithm uses the flow information to ensure that each active flow gets a fair portion of the buffer resources. Flows that monopolize resources receive heavier penalties when packet dropping becomes necessary.

Stabilized RED (SRED) [5] is another flow-based RED approach where the algorithm provides a method to estimate the number of active flows and to identify misbehaving flows without keeping per-flow state information. SRED controls buffer usage by tuning the drop probabilities based on the estimated number of active flows.

2.2. Stochastic Fair BLUE (SFB)

BLUE is an AQM approach that uses packet-loss and link-utilization information instead of average queue length in the congestion-avoidance algorithm [6]. BLUE uses a single probability to drop or mark packets. If the link is idle or the queue becomes empty, BLUE decreases the drop/mark probability. On the other hand, if the queue consistently loses packets due to buffer overflow, BLUE increases the drop/mark probability. As a result of the increased probability, we drop or mark more packets and therefore send out congestion notifications at a higher rate. This adaptive procedure allows BLUE to learn the correct rate for sending congestion signals to the transmitting hosts.

Stochastic Fair BLUE (SFB) uses the BLUE algorithm to protect TCP flows against flows — such as UDP flows — that do not respond to congestion notifications [7]. SFB maintains a small amount of flow-related state information in order to enforce fairness among all the flows. The SFB algorithm quickly drives the drop/mark probability to a very high value, perhaps even one, for an unresponsive flow. In contrast, TCP flows usually maintain low drop/mark probabilities because TCP flows reduce their transmission rates in response to congestion notifications. When the SFB technique identifies an unresponsive flow by observing a high drop/mark probability, SFB applies a bandwidth-limiting policy on that particular flow. The rate-limit policy enforces an allowable amount of data that the flow can enqueue into the buffer, and SFB therefore drops more packets of the unresponsive flows. In simplified terms, the SFB algorithm identifies unresponsive flows and subjects them to rate limiting while allowing responsive TCP flows to perform normally with low drop/mark probabilities.

2.3. GREEN

Generalized Random Early Evasion Network (GREEN) is a proactive queue-management (PQM) method that applies a mathematical model of the steady-state behavior of a TCP connection to drop or mark packets proactively. GREEN attempts to maintain low packet loss and high link utilization while reducing latency and delay jitter. Based on the mathematical model, GREEN is able to give each flow its fair share of bandwidth at the router. The router can use GREEN to identify and police flows that do not respond to congestion notification.

Using the mathematical model that Mathis et al. [12] recommend, Feng et al. [8] derive a mathematical model for a *drop* probability that allows a fair share of bandwidth for every flow. Equation 1 shows that Feng's fair-share drop probability depends on the number of active flows, N, and the round-trip time, RTT. A GREEN router sends out congestion notifications more aggressively when there are more active flows (larger N) or when the round-trip time is shorter (smaller RTT). In the equation, MSS is the maximum segment size, L is the outgoing link throughput of a router, and c is a constant that depends on the acknowledgment strategy (i.e., *delay* or *every packet*).

$$p = \left(\frac{N \times MSS \times c}{L \times RTT} \right)^2 \tag{1}$$

2.4. Explicit Congestion Notification (ECN)

The Internet Engineering Task Force, IETF, has proposed *Explicit Congestion Notification* (ECN) as an alternative to using packet loss for signaling congestion [9]. Instead of dropping packets as congestion increases, routers using this congestion-avoidance algorithm set the congestion-indicator bit in an IP packet to signal congestion. When a receiving device receives a packet with the congestion-indicator bit set, the receiving device uses the transport-level acknowledge message to communicate the congestion indication to the transmitting device. Upon receiving the explicit congestion notification, the sending device decreases its transmission rate temporarily until the traffic condition of the network improves.

Using IP ECN instead of the traditional packet-loss approach can obviously reduce packet loss and thereby improve performance. However, the ECN technique requires explicit support from both communicating devices to be successful, and both devices have to agree to use this scheme. In addition, all of the routers in the entire network must support this method for ECN to be effective. Outdated routers that do not support ECN might drop packets with the ECN bit set, thus causing the IP ECN approach to fail.

2.5. (Pre-) Congestion Notification (PCN)

The Congestion and Precongestion Notification Working Group of IETF has developed an Internet draft for a Pre-Congestion Notification (PCN) architecture [13]. PCN is an architecture for flow admission and/or termination based on (pre-) congestion information that nodes in a Diffserv domain provide. The aim of PCN is to protect the QoS of inelastic flows within the Diffserv domain. The PCN approach gives an “early warning” of potential congestion in the PCN domain before there is any significant congestion.

If the flow rate through a PCN-enabled interior node exceeds the PCN threshold rate, the node marks all packets with PCN threshold-rate markings. If the flow rate exceeds the PCN excess rate, the node marks some packets with PCN excess-rate markings while marking all remaining packets with PCN threshold-rate markings. These (pre-) congestion markings propagate to PCN egress nodes, and the PCN boundary nodes use this (pre-) congestion information to make decisions on flow admission and/or termination. Fig. 1 shows how the PCN admission and termination controls operate in a PCN domain with three encoding states as the rate of PCN traffic increases.

Note that PCN applies to a Diffserv domain with PCN-enabled nodes, so PCN is not a general solution for all environments.

3. Problems with existing approaches

Real-time IP applications, such as VoIP and IP video conferencing, use the *Real-time Transport Protocol* (RTP) to transport real-time media packets, and RTP runs on top of UDP. Unlike TCP, UDP does not support *acknowledge* (ACK) messages in the protocol, nor does UDP implement any transmission-control mechanism to allow the network to alter the transmission rate of a flow.

Lacking ACK messages, an application that uses UDP as the transport protocol has no way to determine when a router drops any of the packets that the application sends. Packet loss is a reliable indication of congestion, but UDP transmitters, being unable to detect packet loss, cannot react to congestion in the network. Even if a transmitter could

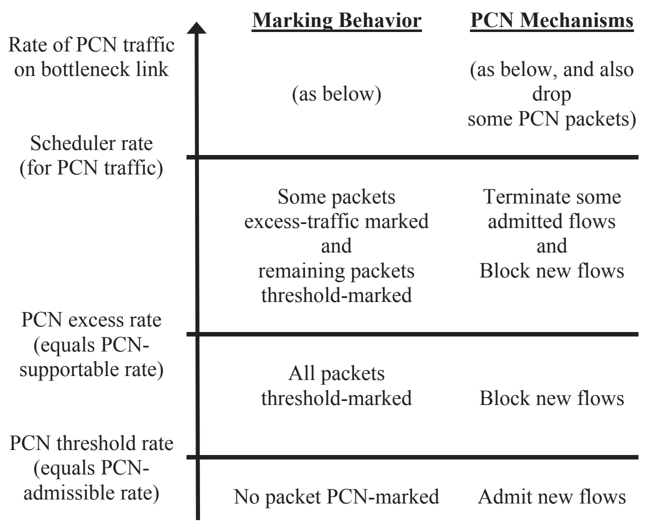
Marking Behavior
PCN Mechanisms
Rate of PCN traffic on bottleneck link
(as below)
(as below, and also drop some PCN packets)
Scheduler rate (for PCN traffic)
Some packets excess-traffic marked and remaining packets threshold-marked
Terminate some admitted flows and Block new flows
PCN excess rate (equals PCN-supportable rate)
All packets threshold-marked
Block new flows
PCN threshold rate (equals PCN-admissible rate)
No packet PCN-marked
Admit new flows

Figure 1. PCN technique

somehow detect the occurrence of packet loss, UDP does not have a mechanism to control its transmission rate. Consequently, UDP applications do not respond well to any of the existing congestion-control algorithms.

Even congestion-avoidance techniques that try to enforce fairness, as flow-based RED and SFB do, are not fully effective against UDP streams, which do not respond to congestion-control mechanisms. Since UDP flows are not responsive to congestion-control signals, these flows simply monopolize the resources of the router in an environment of congestion. Consequently, these streams receive harsh punishment in the form of many lost packets, and they therefore suffer greatly reduced *Quality of Service* (QoS).

TCP-Friendly Rate Adaptation Based on Loss (TRABOL) is an application-level congestion-control algorithm designed specifically for UDP-based applications [14]. TRABOL employs a technique that is similar to the approach of TCP, but TRABOL implements congestion control at the application level while TCP uses the protocol level. Like TCP, TRABOL relies on lost packets to adjust the sender's transmission rate. Unlike TCP, however, TRABOL uses the loss rate computed over a period of time at the receiver side as feedback to tell the sender how to adjust the transmission rate. If the period for calculating the loss rate in TRABOL is too large, the sender could react to the congestion too late or simply adjust the transmission rate incorrectly. If the period for calculating the loss rate in TRABOL is too small, on the other hand, the system could overreact to minor disturbances that do not really indicate congestion. In this case, the flow rate would oscillate needlessly. Furthermore, TRABOL does not address the real-time criterion of real-time UDP-based applications such as VoIP.

The inherent characteristics of real-time applications such as VoIP present additional challenges to the implementation of transmission control. Real-time IP applications typically produce output data at a constant rate, and we need to transport the continuous output stream to the receiving endpoint with minimal delay to maintain the usefulness of the data and a high QoS. Delivering only part of the output stream to reduce the output data rate inevitably degrades QoS. Holding back the real-time data transmission in order to wait for the end of a congestion period increases delay variation (i.e., *jitter*) and lengthens the end-to-end delay, further impairing QoS.

4. Our congestion-control mechanism

We propose a solution that allows real-time IP applications to implement adaptive congestion control in the face of congestion while meeting all of the intrinsic challenges of real-time media systems. Our solution consists of two independent components, congestion detection/notification and adaptive transmission control. Unlike existing approaches that implement congestion control at the protocol level, our technique implements the solution at the application level. As a result, we do not need to change existing protocols or the existing infrastructure to put our approach into practice.

4.1. Congestion detection/notification

Network congestion normally occurs at the routers, and detecting congestion at the router bottleneck is probably the detection approach that provides the most reliability and accuracy. Therefore, the router is the best device for performing congestion detection. Since the router is the first network device that can observe congestion, the router can deliver a notification of congestion sooner than any other device in the network. However, the innumerable routers of the Internet are beyond our control, so deploying new procedures for congestion detection and notification in all the routers of the Internet is impractical.

Our solution employs congestion detection at the receiving endpoint. We merely require both communicating endpoints to agree on the scheme of congestion detection and notification instead of requiring many devices beyond our control to cooperate with our algorithm.

When congestion occurs at the router, the average queue size in the router grows. High queue occupancy at the router increases the transmission delay of the packets since a packet takes more time to work its way through a longer router queue, and packets arrive at the destination later than anticipated. Real-time IP applications commonly transmit data packets at constant intervals, so data packets arrive at the receiving endpoint with consistent periods and minimal variation if the network is idle. Therefore, an increase in the time between the arrivals of consecutive packets at the destination is a good indication of congestion [15].

If congestion is severe enough, a router that implements a congestion-avoidance algorithm, such as RED or its variants, starts to drop packets to curb the congestion. Consequently, data packets disappear from the network, and the receiving endpoint can use packet loss as a clear indication of congestion. If the routers on the transmission path implement IP ECN, the receiving endpoint can recognize the set ECN flag as a sign of congestion.

Statistics that we have collected over a period of several months on the Internet and on a corporate local-area network (LAN) show that even a lightly loaded router experiences sudden microbursts of traffic. These abrupt and brief periods of congestion are severe enough to cause the router to drop packets, but the times between the arrivals of previous consecutive packets do not always show any early sign of congestion. This kind of unforeseen and acute congestion is different from the congestion that builds over a longer period of time, and our congestion-detection algorithm must be able to detect both types of congestion effectively.

The receiving endpoint can recognize slowly building congestion in the network by implementing an algorithm to detect inter-packet delays that are longer than usual. In order to detect variations in inter-packet delays, of course, the receiver must know the arrival rate of the packets. The transmitting party or parties can reveal the transmission-rate information during the negotiation process at the start of a session, or the receiver can quickly and easily calculate the arrival rate of packets after the session starts.

When congestion builds up gradually, the detection algorithm easily detects the pattern of growing inter-packet arrival times before the congestion can cause any noticeable harm. When the filtered arrival time between consecutive packets exceeds a predetermined threshold, the receiver declares a state of congestion.

Unfortunately, there may be no warning sign of growing inter-packet arrival times before a microburst suddenly causes a lost packet, and the inter-packet arrival time would not reveal this problem until the arrival of the next packet after the lost packet(s). Therefore, the detection algorithm employs a time-out procedure to detect a delayed or missing packet immediately. The time-out mechanism declares a congestion condition when a packet has not arrived at the receiver by some predetermined time after the estimated packet-arrival time but before the estimated arrival time of the subsequent packet. The time-out procedure detects a congestion condition well ahead of the transmission time of the subsequent packet, so the transmitter can delay transmission briefly to avoid losing more packets. With the inclusion of the time-out feature, the algorithm detects sudden microbursts as well as slowly growing episodes of congestion.

Additionally, the system uses packet sequence numbers to detect out-of-order packets and considers an out-of-order packet to be an indication of a lost packet even though the "lost" packet may arrive later or may have previously arrived out of order. When packets are out of order, we certainly have congestion.

Upon detecting congestion, the receiver sends the transmitter a congestion notification containing an estimate of the severity of the congestion. This notification tells the transmitter that congestion is present and to what degree congestion is present. The receiver can piggyback the notification message with the next data packet that the receiver transmits to the sender, thereby avoiding the undesirable effect of adding notification packets to a network that is already congested. Both ends of a VoIP system continually send packets to each other at brief intervals, typically twenty milliseconds, so timely notification without any added packets is entirely feasible.

Our extensive study of the characteristics of packet-switched network traffic shows that network congestion is direction dependent. In other words, a congestion condition on the path from A to B does not necessarily imply a similar congestion condition on the path from B to A. In fact, we often observe relatively idle traffic on the opposite direction of a congested network path. Therefore, sending a congestion notification from the receiver to the transmitter typically does not elevate the severity of the congestion.

Nevertheless, the network may drop the packet containing the congestion notification, so the receiver should send multiple congestion notifications. However, each of the notification packets for the same occurrence of congestion must contain the same unique identifier so the sender reacts only once to the first notice it receives and ignores the rest. The receiver can stop transmitting the notification messages when it observes a lower transmission rate from the sender, or the receiver can stop transmitting the notification messages after some duration, especially when the congestion subsides.

4.2. Our implementation of congestion detection

Our congestion-detection algorithm measures and evaluates the inter-packet arriving intervals, so the algorithm not only detects congestion in the network but also estimates the severity of the congestion. Additionally, our algorithm anticipates future congestion that is likely to occur soon after an episode that is part of a longer period of network congestion. Our detection procedure computes the absolute value of the difference between the inter-packet arriving interval and the original inter-packet transmission interval. Then we pass the resultant value through a simple first-order infinite impulse response (IIR) filter to obtain the severity of the congestion in the network.

Equation 2 shows the first-order IIR filter that our algorithm uses. Y_n is the current output value of the filter, and it is the estimate of the severity of the congestion in the net-

```
CurrentTime = GetCurrentTime();
Xn = ABS((CurrentTime -
          PrevPacketArrTime) -
         PacketTransmissionInterval);
if (NOT Timeout)
  { PrevPacketArrTime = CurrentTime; }
if (LostPacket OR Timeout)
  { Xn = MIN(Xn, MAX_Xn_LIMIT);
    Xn = MAX(Xn, Yprevious); }
if (Xn >= Yprevious)
  { C1 = 0.9;
    C2 = 0.1; }
else
  { C1 = 0.03;
    C2 = 0.97; }
Yn = (C1 * Xn) + (C2 * Yprevious);
if (Yn >= CONGESTION_THRESHOLD)
  { DeclareCongestion(Yn); }
Yprevious = Yn;
```

Figure 2. Pseudo code of congestion-detection algorithm

work. Y_{n-1} is the previous output value of the filter, so it provides feedback. X_n is the current input sample to the filter, and it is the absolute value of the difference between the current inter-packet arriving interval and the inter-packet transmission interval. C_1 and C_2 are coefficients of the IIR filter.

$$Y_n = (C_1 \times X_n) + (C_2 \times Y_{n-1}) \tag{2}$$

Fig. 2 shows an excerpt from the pseudo code for our congestion-detection algorithm [16]. Our implementation uses an IIR filter with fast-rise and slow-decay characteristics. The fast-rise characteristic of the filter allows the measurement of congestion severity to increase quickly when congestion builds up. The slow-decay characteristic of the filter allows the congestion-severity value to fall gradually after the network congestion subsides, thus anticipating the likely prospect that network congestion might persist or that additional network congestion might occur shortly after the current episode of congestion. In comparison to an episode of moderate congestion, a period of severe congestion raises the measurement of congestion severity to a higher value, and a higher congestion-severity value requires a longer time to decay to a level below the congestion threshold. We can use different C_1 and C_2 coefficients to adjust the rise rate and decay rate of the IIR filter if necessary. A slower decay rate allows the system to anticipate congestion that might occur further into the future.

Fig. 3 illustrates inter-packet arriving intervals that are typical for data that we captured from the Internet during highly congested periods. The transmitting endpoint was transmitting one VoIP packet every twenty milliseconds in this experiment, and the receiving endpoint observed erratic inter-packet arriving intervals ranging from zero milliseconds (i.e., two or more packets arriving at about the same time) to ninety milliseconds or more during this period of severe congestion. The receiving endpoint even observed lost packets and, in some cases, out-of-order packets.

Fig. 4 illustrates the output from our congestion-detection mechanism for the inter-packet arriving intervals of Fig. 3. As we can see in Fig. 3, the receiving endpoint repeatedly observed extreme delays in the arriving packets. These extreme delays produce significant congestion-severity values that remain above the congestion threshold throughout the slow decay of the IIR filter. Our congestion-detection algorithm therefore reports that congestion continues throughout the entire episode of network congestion.

Fig. 5 shows inter-packet arriving intervals that we captured from a corporate LAN. As with the previous experiment, the transmitting endpoint transmits a VoIP packet every twenty milliseconds. During this period, the receiving endpoint observed three episodes of mild to serious congestion. Using the data in Fig. 5, our congestion-detection algorithm generates the output that appears in Fig. 6.

4.3. Adaptive transmission control

The real-time application at the transmitting endpoint implements the adaptive transmission-control mechanism. Upon receiving a congestion notice from the endpoint at the receiving end of the transmission, the transmitting end-

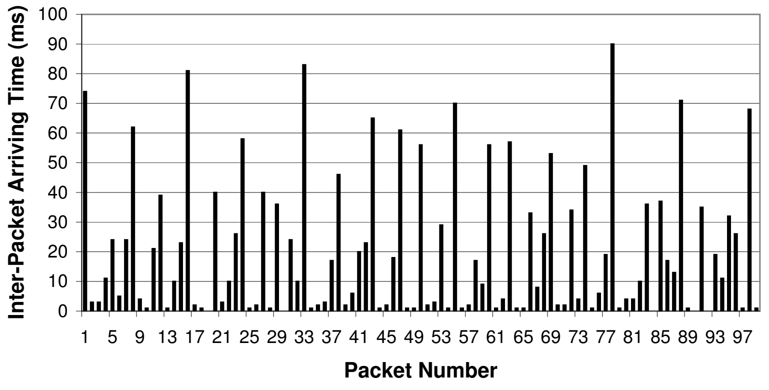

Figure 3. Inter-packet arriving interval (Internet)

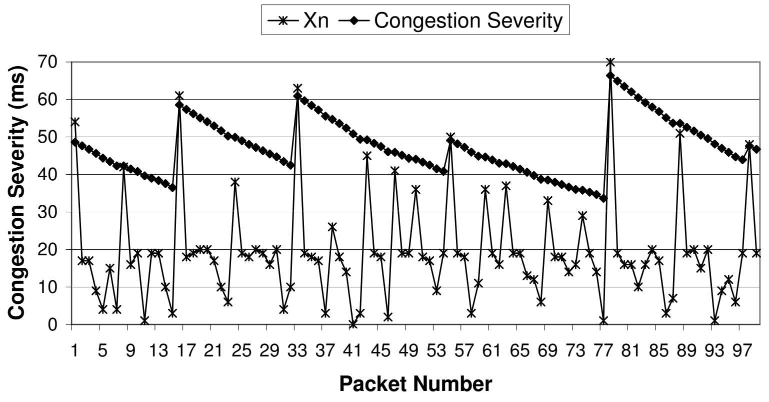

Figure 4. Estimate of congestion severity (Internet)

point lowers its transmission rate to reduce bandwidth consumption. This reduction of the transmission rate is not as simple for a real-time application such as VoIP as it is for a TCP application, though, because the transmitting endpoint must deliver all of the real-time data with minimal delay to maintain a high QoS.

One straightforward approach for reducing bandwidth consumption is to switch to a different compression algorithm that can compress the real-time data to a greater degree and thereby produce an output stream with a lower bit rate. Generally, real-time VoIP applications already use compression algorithms to compress real-time data before sending the data across the network because uncompressed voice data typically consumes too much bandwidth. The transmitting endpoint can further reduce bandwidth consumption by switching to a different compression algorithm that achieves a lower bit rate. The greater compression typically results in a slight QoS penalty, but this penalty is mild in comparison to the extreme QoS penalty that occurs as a result of the packet loss that normally stems from network congestion. This compression-switching approach requires both communicating endpoints to support the same set of compression schemes.

Various alternative data-compression algorithms have differing bandwidth requirements, and some compression algorithms support multiple compression ratios. For example, the ITU-T (International Telecommunication Union Standardization Sector) standard G.729 compresses audio data to 8 kbps [17], G.729 with annex D compresses audio data to 6.4 kbps, and G.729 with annex E compresses audio data to 11.8 kbps. The ETSI (European Telecommunications Standards Institute) GSM 06.90 standard, GSM adaptive multi-rate (GSM-AMR), supports multiple bit rates of 4.75 kbps, 5.15 kbps, 5.9 kbps, 6.7 kbps, 7.4 kbps, 7.95 kbps, 10.2 kbps, and 12.2 kbps [18]. A transmitter can reduce bandwidth by simply switching to a different compression algorithm in the same family.

Another method for reducing bandwidth usage is to transmit the same data with fewer transmissions, thus reducing packet-header overhead. The sending endpoint achieves

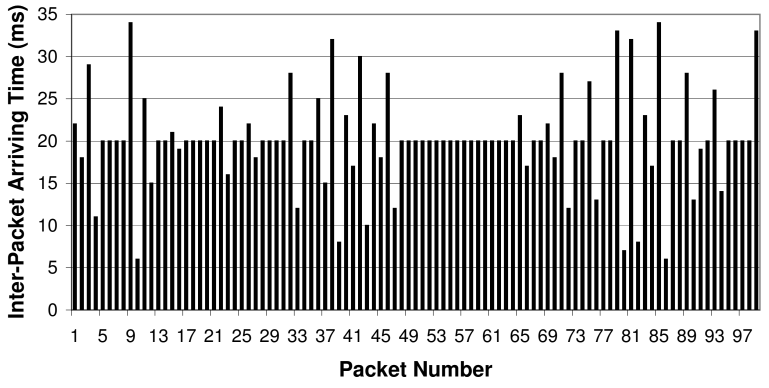

Figure 5. Inter-packet arriving interval (LAN)

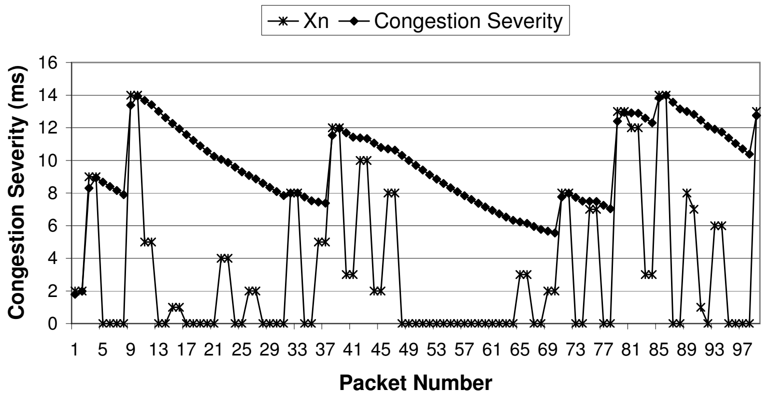

Figure 6. Estimate of congestion severity (LAN)

this goal by covering a longer period of time with the data that it packs into each IP packet. For example, a VoIP endpoint that transmits twenty milliseconds of audio data per IP packet can cut the number of transmissions by a factor of two if the endpoint sends forty milliseconds of audio data per IP packet. This approach significantly lowers bandwidth consumption by reducing the amount of bandwidth that the transmitter consumes with packet-header overhead. Obviously, this technique achieves its goal at the cost of adding a slight delay to the real-time data stream, but this approach does not trim or further compress any audio data.

A VoIP endpoint that packages twenty milliseconds of G.729 compressed audio per IP packet requires 24.0 kbps, including the overhead of the IPv4, UDP, and RTP headers at forty bytes per packet. If the VoIP endpoint packages forty milliseconds of G.729 compressed audio per IP packet, the required bandwidth diminishes to only 16.0 kbps, including header overhead. This simple scheme results in a tremendous bandwidth saving of 8.0 kbps or 33%. This approach does introduce an additional twenty milliseconds of delay into the audio stream, of course, but the effect of this small added delay on QoS is typically insignificant [19]. In fact, this method can actually *reduce* the overall delay because the technique alleviates queue delays in the routers, often more than compensating for the small delay between consecutive transmissions. Table 1 shows the bandwidth consumptions and savings with different amounts of compressed G.729 audio data in each IP packet.

The transmitting endpoint can simultaneously employ *both* of the bandwidth-reduction techniques that we have proposed since the two methods are independent of each other. The transmitting device can switch to a compression algorithm that produces a lower bit rate, and the transmitter

Table 1. G.729 Bandwidth utilization

Audio Length (milliseconds)	Bandwidth (kbps)	Packets per Second	% Savings
10	40.000	100.00	-
20	24.000	50.00	40.00%
30	18.667	33.33	53.33%
40	16.000	25.00	60.00%
50	14.400	20.00	64.00%

can concurrently package more data into each IP packet to lower the number of transmissions and save bandwidth on the packet-header overhead.

The congestion-severity information in the congestion notification that the receiver sends to the transmitter allows the transmitter to gauge its response according to the severity of the congestion. Using the congestion-severity information, the transmitting endpoint selects the bandwidth-reduction method that is most appropriate for the level of congestion, thereby maintaining optimal link utilization and throughput while simultaneously curbing congestion.

When network congestion recedes, the transmitting endpoint adjusts its transmission rate back to the original setting. The receiving endpoint detects the improvement in the congestion, and the receiving endpoint conveys this information to the transmitting endpoint. After learning that the traffic condition has improved, the transmitting endpoint increases its transmission rate to reduce delay and improve the grade of service. The transmitting endpoint can alternatively operate with the optimized transmission procedure for a predetermined period of time. When the fixed duration for using the lower transmission rate expires, the transmitting endpoint automatically resets its transmission rate.

5. Measure of improvement

To illustrate the effectiveness of our technique, let us examine the performance improvement that we can achieve. Consider a scenario in which the router throughput is 10.0 Mbps and we have 700 VoIP streams going through the router. If we use G.729 to compress the audio data and pack 20 milliseconds of audio into each IP packet, we require 16.80 Mbps of bandwidth, 6.80 Mbps more than the router can handle. As a result, the router drops an average of 20.238 packets per second for each flow. That drop rate translates into more than 400 milliseconds of lost audio data in each second for each flow, a devastating loss of more than 40%!

If the VoIP applications employ our solution and begin to package 40 milliseconds of audio data into every IP packet, we require only 11.20 Mbps for all 700 audio streams combined. That bandwidth reduction cuts the drop rate to 2.679 packets per second per flow for an average of about 107 milliseconds of audio loss in a second for each flow, a tolerable loss rate of just 10.7%. If the VoIP applications switch to use G.729 with annex D and also pack 40 milliseconds of audio per IP packet, the required bandwidth decreases to 10.08 Mbps. In this case, each audio stream loses an average of only 0.198 packets per second for a loss rate of just 0.79%, almost *zero* loss! Fig. 7 illustrates the loss comparison for the case that we have examined.

6. Delay versus packet loss

Since one aspect of our adaptive transmission control increases the overall delay by using larger, less-frequent packets, we must consider the potential negative impact of increasing the delay versus the positive impact of reduced packet loss.

We can use the ITU-T G.107 E-Model to analyze the QoS impact of the additional delay that our transmission-control method introduces into the audio stream. The E-model is an analytical model that evaluates the conversational quality of a telephony system. The E-model includes many items (e.g., room noise, echo, and circuit noise) that are independent of both packet loss and delay, but we isolate the effect of delay and thereby determine the quality differences that are due to the delay variations.

When the total delay does not exceed 100 milliseconds, the E-model indicates that there is *no degradation at all* due to the delay. This case is the relevant case for most VoIP systems since designers try to make the delays low enough to eliminate or at least minimize the effects of delays. The delay degradations remain insignificant until the overall delay reaches a level of about 200 milliseconds, and the degradation grows as the delay approaches the talker-overlap threshold of 250 milliseconds. The worst case occurs when an added delay of 20 or 40 milliseconds above the base delay pushes the total delay beyond the talker-overlap threshold, in which case the MOS rating degrades

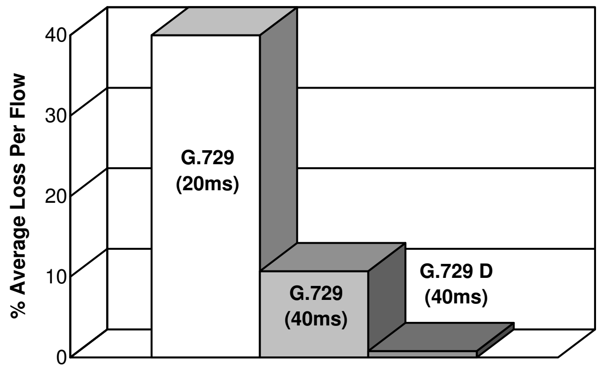

Figure 7. Loss-analysis case study

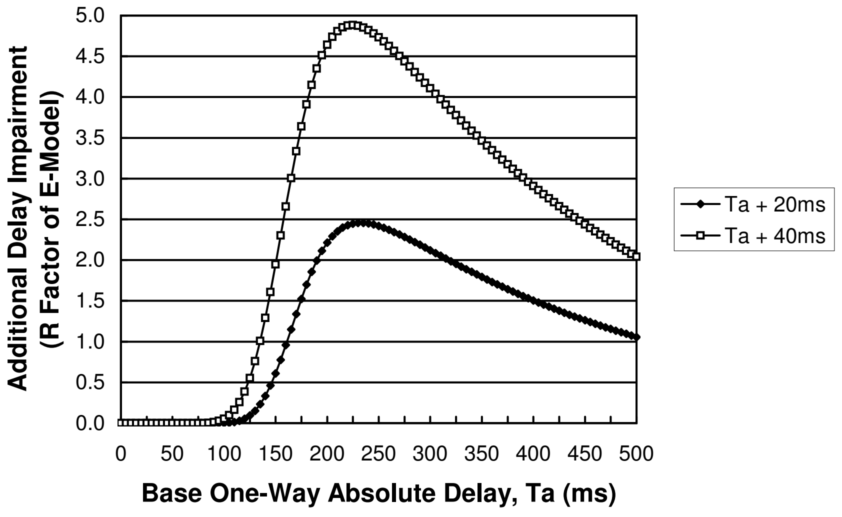

Figure 8. E-model evaluation of delay impairment

by approximately 0.1 for an added delay of 20 milliseconds or by 0.2 for an added delay of 40 milliseconds. This worst-case situation is not important for practical applications, though, because any system that is close to the talker-overlap threshold is already a marginal system for VoIP.

Fig. 8 illustrates the added delay impairment — in units of the R-factor of the E-Model — that results from the introduction of delay increases of 20 milliseconds and 40 milliseconds. Based on the E-model, the increase of 20 milliseconds in the delay in the audio stream typically has *zero* impairment to at most 2.5 units of R-factor — about 0.1 in Mean Opinion Score (MOS) — of impairment in the QoS.

In our research on packet loss in a real network, we evaluated the MOS ratings of G.729 streams transmitting at a 20-millisecond interval in a simulated real-network environment. Our study showed that a difference of 30% (e.g., from 10% to 40%) in the loss of audio data translates to a difference in QoS impairment of more than one full unit on the MOS scale [19]. Fig. 9 illustrates the MOS ratings of a G.729 stream with various degrees of packet loss in a real network.

Our results show that adding a small delay to reduce packet loss is clearly a good tradeoff. The QoS degradation from the added delay is typically zero or at most minuscule while the resulting QoS improvement from the reduced packed loss is significant.

7. Conclusion

Congestion control and congestion avoidance are popular research topics, so investigators have done considerable work in these areas. However, most of the well-known congestion-avoidance techniques exploit the transmission-control mechanism in TCP. Therefore, these approaches are

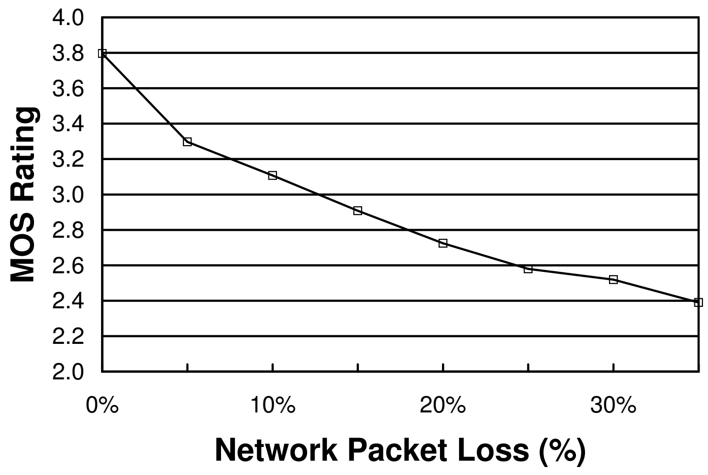

Figure 9. MOS ratings with packet loss

not effective for real-time IP applications, which typically use UDP as the transport protocol. The intrinsic characteristics of real-time media applications pose additional inherent problems to the already-challenging congestion-control issue.

Our new technique for congestion detection and adaptive transmission control for real-time IP applications such as VoIP is an application-level approach. For that reason, we can implement our solution on the current infrastructure using existing protocols. Our method requires only simple upgrades to the implementations of the transmitting and receiving endpoints. As the network becomes congested, the communicating endpoints using our method reduce bandwidth utilization to adapt to the congested environment. When the congestion subsides, the endpoints adapt to the improvement in the traffic condition and return to their original transmission settings.

Since our method is an application-level approach that runs on the endpoints, the implementation is fully scalable with respect to the size of the network and the number of flows. Each endpoint performs a simple task that demands very little in terms of processing power or memory. Together, all of the endpoints in the system cooperate to alleviate the problems that congestion poses for real-time applications such as VoIP.

In contrast to existing congestion-avoidance techniques, which simply identify and impose heavy penalties on unresponsive real-time flows, our technique cooperatively lessens the bandwidth consumption of these flows by lowering their transmission rates. As a result, real-time IP applications that use our approach experience fewer lost packets and achieve higher QoS.

References

[1] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August 1993, pp. 397–413.

[2] Wu-Chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin, "A Self-Configuring RED Gateway," *Proc. IEEE Conference on Computer Communications, INFOCOM 1999*, New York, New York, March 1999, pp. 1320–1328.

[3] Eric Horlait and Nicolas Rouhana, "Dynamic Congestion Avoidance Using Multi-Agent Systems," *Proc. Mobile Agents For Telecommunication Applications, MATA 2001*, Montréal, Canada, August 2001, pp. 1–10.

[4] Dong Lin and Robert Morris, "Dynamics of Random Early Detection," *Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM 1997*, Cannes, France, September 1997, pp. 127–137.

[5] T. Ott, T. Lakshman, and L. Wong, "SRED: Stabilized RED," *Proc. IEEE Conference on Computer Communications, INFOCOM 1999*, New York, New York, March 1999, pp. 1346–1355.

[6] Wu-Chang Feng, K. Shin, D. Kandlur, and D. Saha, "The BLUE Active Queue Management Algorithms," *IEEE/ACM Transactions on Networking*, Vol. 10, No. 4, August 2002, pp. 513–528.

[7] Wu-Chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin, "Stochastic Fair BLUE: A Queue Management Algorithm for Enforcing Fairness," *Proc. IEEE Conference on Computer Communications, INFOCOM 2001*, Anchorage, Alaska, April 2001, pp. 1520–1529.

[8] Wu-Chun Feng, Apu Kapadia, and Sunil Thulasidasan, "GREEN: Proactive Queue Management over a Best-Effort Network," *IEEE Global Telecommunications, GLOBECOM 2002*, Vol. 21, No. 1, November 2002, pp. 1784–1788.

[9] IETF RFC-2481, A Proposal to Add Explicit Congestion Notification (ECN) to IP, January 1999.

[10] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons Not to Deploy RED," *Proc. International Workshop on Quality of Service, IWQoS 1999*, London, UK, June 1999, pp. 260–262.

[11] Vishal Misra, Wei-Bo Gong, and Donald Towsley, "Fluid-Based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," *Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM 2000*, Stockholm, Sweden, August 2000, pp. 151–160.

[12] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM Computer Communication Review*, Vol. 27, No. 3, July 1997, pp. 67–82.

[13] IETF Internet draft, Pre-Congestion Notification (PCN) Architecture, October 2008.

[14] S. Bangolae, A. Jayasumana, and V. Chandrasekar, "TCP-Friendly Congestion Control Mechanism for a UDP-Based High-Speed Radar Application and Characterization of Fairness," *Proc. IEEE Communication Systems, ICCS 2002*, Singapore, November 2002, pp. 164–168.

[15] P. Maryni and F. Davoli, "Load Estimation and Control in Best-Effort Network Domains," *Journal of Network and Systems Management*, Volume 8, Issue 4, December 2000, pp. 527–541.

[16] T. Chua and D. Pheanis, "Application-Level Adaptive Congestion Detection and Control for VoIP," *Proc. IARIA International Conference on Networking and Services, ICNS 2007*, Athens, Greece, June 2007, pp. 84–89.

[17] ITU-T Recommendation G.729, Coding of Speech at 8 kbit/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP), March 1996.

[18] ETSI GSM 06.90, Digital Cellular Telecommunications System (Phase 2+); Adaptive Multi-Rate (AMR) Speech Transcoding, 1998.

[19] T. Chua and D. Pheanis, "QoS Evaluation of Sender-Based Loss-Recovery Techniques for VoIP," *IEEE Network*, Vol. 20, No. 6, November/December 2006, pp. 14–22.