# Service Guarantees in Deflection Networks

Wladek Olesinski & Pawel Gburzynski
Department of Computing Science
615 GSB University of Alberta
Edmonton, Alberta, CANADA T6G 2H1
e-mail: [wladek,pawel]@cs.ualberta.ca

## Abstract

*We*[1] *propose a mechanism for providing long term quality of service in deflection networks. Our proposed protocol is in fact a* call admission control *(CAC) scheme based on session probing combined with a feedback technique that allows a station originating a new session to notify other stations about its bandwidth requirements. This CAC scheme works in an environment in which there are no explicit connections or calls. No resources are reserved in advance and no bandwidth is wasted on passing tokens, which means that our protocol is easily scalable. With the increasing size and/or capacity of the network, only the normalized duration of the setup phase has to be extended. This setup phase of our protocol loosely corresponds to a connection setup phase in a connection-oriented store-and-forward network, e.g., ATM. However, we do not set up actual connections: all packets in the underlying network are routed according to the standard and simple deflection rules.*

## 1. Introduction

We consider *pure deflection networks*, i.e., ones that never lose packets because of a limited buffer space at an intermediate switch. Packets that at the moment of arrival cannot be relayed via their preferred routes (because those routes are busy) are deflected, i.e., relayed via suboptimal routes. This concept, traditionally illustrated by the Manhattan-street networks (MSN) introduced and analyzed by Maxemchuk in [9, 10, 11], works under the assumption that the amount of buffer space available at a switch is sufficient to determine the route preference of an incoming packet before the packet is relayed via one of the outgoing links. No buffer space is required for storing packets that cannot be immediately relayed via their optimal routes, although it may make sense to store such packets temporarily before deflecting them right away [10].

Because of deflection, packets belonging to the same session may travel different paths and may arrive at the destination out of order. This property of deflection networks has been traditionally perceived as a serious disadvantage compromising their reliability in real-time applications, in which the timing of arriving packets is important (e.g., voice and video). This presumption has resulted in an almost total neglection of deflection networks in contemporary networking and has brought us the connection-oriented paradigm of ATM as the only solution to be adopted by the serious networks of the future.

But (as one of us argued in [1]) switched networks are haunted by other problems that are absent in deflection networks. First, a switch insisting on relaying sessions via fixed routes has to deal with the problem of limited buffer space. Accepting only as much traffic as to always be able to guarantee the desired *quality of service* (QoS) would result in a very inefficient utilization of network resources. Second, foolproof techniques of dynamic bandwidth negotiation involve feedback across the network and they become useless if the network is large and/or fast.

Consequently, because of the statistical multiplexing and unpredictability of real-life traffic scenarios, switched networks operating under their intended loads are bound to lose some packets. In a deflection network, the problem of packet misordering is solved at the destinations by using *reassembly buffers* [5]. Owing to the unpredictability of routes, a packet can be lost, if it happens to arrive so late that the reassembly buffer can no longer help. Thus we are confronting two approaches: one in which packets can be lost at all intermediate switches-relays because of the limited buffer space, and one in which packets can be lost at the destination for essentially the same reason. Thus, at this level, there is no significant difference between the two approaches.

One advantage of deflection networks over store-and-

[1]The current address of the first author is Alcatel, 600 March Road, Ottawa, Ontario, Canada K2K 2E6.

forward networks is that a destination in a deflection network allocates the reassembly space on a per session basis; thus, buffers are only used when required by the session. With this approach, the destination (more knowledgeable about the specifics of its sessions than an intermediate switch trying to accommodate everybody) can "know better" how much buffer space is needed to accommodate its needs. It can also allocate more buffer space dynamically, e.g., if it turns out to be insufficient, and the trade-off criteria for buffer allocation are clear and local (they don't affect sessions belonging to other parties).

Long, sustained, real-time, isochronous transfers require packets to arrive in order (e.g., video, voice). A solution to the problem of maintaining QoS guarantees must address this scenario. First, the network may be heavily loaded when a source intends to initiate a new session. In such a case, if the QoS requirements (e.g., packet loss and end-to-end delay) cannot be met with the present network load, the source should be notified about this fact and the session (call) should be blocked. Second, it may happen that an already active session (a call in progress)[2] is disturbed by increased load in the network, caused by new sessions or by a changed activity pattern of the existing sessions.

Several techniques of requesting/reserving bandwidth in deflection networks have been proposed in the literature. For example, backlogged switches may signal their needs using a dedicated field in the packet header [4]. Another solution [8] is based on imposing a virtual multi-ring topology onto the physical (mesh) topology and using a fair bandwidth allocation scheme suitable for rings. However, it assumes that the network is in fact an MSN, i.e., its physical topology is that of a torus and its connectivity is 2.

The common problem of all "obsessive" feedback-based solutions is a poor scalability to high transmission rates and/or large geographical ranges. Such techniques tend to waste bandwidth in proportion to the bandwidth-delay product of the network.

Our proposed mechanism offers a CAC procedure that determines whether a new session can be admitted to the network. It also allows an already initiated session to sustain its QoS regardless of other activities in the network. Although the amount of time needed by the CAC algorithm to decide whether an isochronous session should be admitted or blocked depends on the bandwidth-delay product of the network, this algorithm is only executed at the beginning of the session. Once the session has been admitted, no further negotiations are needed before packet transmission. In response to bandwidth requests from other switches, a source may adjust its *utilization rate* of some outgoing links, but this operation does not disturb the sessions in progress. Moreover, datagram traffic is not subjected to any bandwidth negotiations: it can be expedited spontaneously at any time as long as its transmission does not violate the current utilization rates of the outgoing links.

To illustrate our approach, we focus on a videophone application, although the proposed protocol can be easily extended to accommodate other isochronous applications like videoconferencing and transmitting movies.

[2] We avoid the word "call" in this context, and say "session" instead, as the former suggests the presence of a connection (i.e., preallocated path) in the network. A session in a deflection network is a logical sequence of messages to be exchanged between a selected source-destination pair.

## 2. Network model

A deflection network consists of a number of switches interconnected into a grid (e.g., see Figure 1). Every switch has the same connectivity $k$, which determines the number of outgoing and incoming links.

The network operates in a slotted manner, in a way similar to MSN [10]. In one routing cycle, the switch accepts incoming slots from all its input ports and routes them to the output ports. If an incoming slot is nonempty and addressed to the current switch, the switch receives the contents of the slot and marks it as empty. If an incoming slot is empty, the switch is free to fill it with its own outgoing packet. The routing strategy assigns output ports to all incoming slots that appear nonempty after the above operations. Packets are assigned to outgoing links such that the sum of the shortest distances to their destinations is minimized. If several possible selections give the same minimum sum, one of them is made randomly. We say that a routing scheme with these properties is *locally optimal* [3].

In our model, we assume that slots are never buffered at a switch, except for the alignment and routing. Every switch is equipped with a fractal (self-similar) traffic generator [2, 7, 16]. A single session is described by the Hurst parameter $H$, the number $M$ of independent and probabilistically identical fractal renewal processes (FRPs), and the load of $\lambda$ kB per frame. To simplify the discussion, we assume that a single switch can only handle one session at a time, but our protocol does not require this restriction.

Packets generated by a switch (or a host attached to it) are stored in a queue from which they are removed and inserted into the network at some maximum rate determined by the currently prescribed *utilization rates* of the outgoing links. By adjusting the link utilization rates, we control the amount of traffic contributed by the switch to different regions of its neighbourhood.

At startup, the utilization rate of every outgoing link is set to 1. That is, the source is allowed to insert its packets to the network in every available free slot. A switch may transmit more than one packet during one slot time (up to $k$ packets), if more than one outgoing slot happens to be empty.

The recipient of an isochronous session allocates a playout (reassembly) buffer of size $B$ packets. At the beginning, the playout buffer is empty. When the number of packets in the buffer reaches $B_R = R * B$, where $0 < R < 1$, the recipient starts removing (receiving) packets at the rate required by the session. Packets that arrive too late to be played out are dropped, and so are packets that arrive while the buffer is full. The role of $R$ is to provide an initial backlog of packets to be received, to compensate for the variability and unpredictability of network delays. The impact of $R$ on packet loss and jitter is discussed in [15].

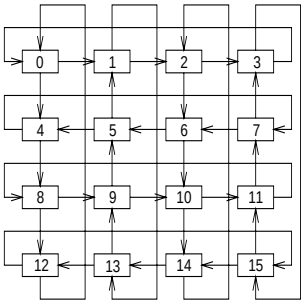



**Figure 1. Torus, size 16, connectivity 2**

Our proposed bandwidth allocation scheme is independent of the network topology and connectivity. For experiments, we have considered regular and irregular network topologies with connectivities 2 and 4. For the sake of brevity, we present the results obtained for the 2-connected torus (see Figure 1).

## 3. The protocol

Let us consider a videophone session. One property of such a session is its symmetry, i.e., there are two parties $S$ and $\hat{S}$ that must both act as a transmitter and receiver.

Before $S$ and $\hat{S}$ start an "actual" session, they probe the network by transmitting dummy packets to each other. This traffic must be representative of the actual session, as it is going to look when admitted to the network. During this probing time, the two sources determine if the network is capable of sustaining their session with the desired QoS. This phase of the transmitter's protocol is called the *setup phase*.

The setup phase completes when the two sources have found out whether or not their probe session disturbs the already active sessions, and the QoS perceived by the receivers at $S$ and $\hat{S}$ is acceptable. Upon a successful completion of the setup phase, the transmitter will begin the normal session, i.e., the *active phase*. If the setup phase completes unsuccessfully, indicating that the session cannot be admitted at the desired QoS without bringing the sessions in progress below their QoS requirements, the new session is blocked.

### 3.1. The active phase

A video source typically generates at least 20 frames of packets per second. The packets of every frame are put into a queue from which they are extracted at the maximum possible rate limited by the availability of empty slots and the current utilization rates of the outgoing links.

Consider a source (i.e., host) $S$. Every source monitors the traffic and records certain information, for example, sources of passing packets. This information is later used by the protocol. Let $F_Q$ denote the length of the so-called *observation period*, expressed in frames, used to assess the stability of the packet queue at $S$. Before generating a new frame number $i$, the source stores the current size of the packet queue $Q_i$. Having generated $F_Q$ frames, the source examines the last $F_Q$ values of $Q_i$.

Let $V_g$ denote the number of times when $Q_{i+1} > Q_i$, and $V_l = F_Q - V_g$ show how many times $Q_{i+1} \leq Q_i$ for $i = 0, \ldots, F_Q - 1$, where $Q_i$ is the size of the queue immediately before frame $i$ is generated. The queue is considered to be growing within the observation period $n$, if $V_g^n \geq V_l^n$ and $Q^n_{F_Q-1} > Q^{n-1}_{F_Q-1}$. In other words, the queue is assumed to be growing if its size monitored at the beginning of a new frame is usually greater than it was at the beginning of the previous frame, and the queue size at the end of the current observation period is greater than it was at the end of the previous observation period. For example, the sequence of $< 0, 30, 25, 18, 15 >$ will not indicate a growth. This sequence may represent a temporary increase of the queue size caused by a long burst of packets generated by the source.

If the queue is growing, it is likely that packets coming from other sources have been inhibiting the source $S$. In such a case, $S$ selects a setup source $K$ (i.e., a source in the setup phase) such that

$$V = M_K^l * \frac{1}{r_l + 1} \tag{1}$$

is maximized. $M_K^l$ is the average number of packets per frame time sent by $K$ through its link $l$ that have been seen by $S$. $1/r_l$ is the utilization rate of link $l$ at $K$. Note that $S$ finds the identity of $K$ in its internal structures, created while monitoring the traffic.

To make sure that $S$ and other switches are able to collect these statistics, the values of $r_l$ and $l$, as well as the status (phase) of the source must be carried in all transmitted packets. The overhead for this is very low, because each of those attributes can be efficiently represented within a few bits.

If $K$ lowers the utilization rate of $l$ from $1/r_l$ to $1/(r_l + 1)$, formula 1 will approximate the factor by which the number of packets transmitted by $K$ through link $l$ will drop. Packets transmitted through this link often pass through switch $S$ and that is why $S$ will try to slow down $K$'s transmission over $l$.

Next, $S$ checks if the condition

$$\gamma_Q < V \tag{2}$$

holds, where $\gamma_Q$ is the growth factor of the packet queue at $S$ expressed as the number of packets per frame, averaged over the last $F_Q$ frames, by which the queue has grown. If (2) holds, then, assuming a similar intensity and pattern of the traffic originating at the setup source $K$ in the future, the decrease of the number of packets transmitted by $K$ through link $l$ should stop the growth of the packet queue at $S$.

It may happen that (2) does not hold for the single value of $M_K^l$ selected in the previous step. This suggests that slowing down only one source will not help. In this case, $S$ will try to find (in its internal structures) another source $K$ with a large value of $V$. The new value of $V$ will be added to the previous one, and $S$ will re-examine (2).

If (2) holds, $S$ will send a *slow-down* (SD) request to source $K$. Note that if multiple sources are selected for a slow-down, the SD request can be sent as a multicast packet. In [13] we have proposed a few simple and efficient methods of implementing multicast traffic in deflection networks. Following the transmission of the SD request, the source will refrain from transmitting more SD messages (even if its queue continues to grow) for the time of 3 frames. As we explain in [12], this time is sufficient to make sure that a slowed-down source $K$ has had enough time to respond. Using the approach described above, $S$ will keep sending SD messages to setup sources until its packet queue ceases to grow.

If (according to formula 2) the queue at $S$ is likely to grow even if all sources registered during the last $F_Q$ frames agree to slow down, $S$ may send SD to some sources several times. It is thus possible that a given setup source will decrease its transmission rate over some link more than once upon request of a single source. If a setup source is unable to do so without maintaining the stability of its own packet queue, its session will be blocked.

When a source has nothing more to transmit, it sends *connection finished* (*CF*) message to the receiver.

## 3.2. The setup phase

In this phase, source $S$ and its peer $\hat{S}$ send "dummy" packets trying to determine if the tentative session between them is likely to sustain the required QoS. For some time, $S$ monitors the behaviour of its queue. If the queue grows, it usually means that the network load is high and $S$ is often inhibited by packets coming from other sources. In such a case, the session being initiated by $S$ is blocked. The session is also blocked when the peer $\hat{S}$ is blocked, when the QoS, as perceived by the receiver at $\hat{S}$ is not fulfilled (see Section 3.3), or when the source receives a slow-down SD request (see Section 3.1) with which it cannot comply.

First, let us see how the setup source $S$ responds to a slow-down request. When it receives an SD message from switch $K$, $S$ checks if

$$\tau + \frac{1}{r_l} * \tau_l < N_F \tag{3}$$

$1/r_l$ is the utilization rate of link $l$ at $S$. The purpose of the SD request sent by $K$ is to reduce this utilization rate. $\tau$ is the time interval between the moment of transmission of the last packet from a frame and the moment when the frame was generated. This value is averaged over the last $F_Q$ frames. $\tau_l$ has the same meaning as $\tau$ except that we look at those packets of their respective frames that were the last packets transmitted through link $l$. $N_F$ is the time needed to pass one frame of packets.

If $S$ reduces the utilization rate of $l$ from $1/r_l$ to $1/(r_l + 1)$, it will take on average $\frac{r_l+1}{r_l} * \tau_l - \tau_l = \frac{1}{r_l} * \tau_l$ more time to send the same number of packets through link $l$. When this value is added to $\tau$, the result stands for the overall time needed by $S$ to send all the generated packets. If this value exceeds $N_F$, $S$'s queue will be likely to grow. In that case, the session initiated by $S$ will be blocked (Section 3.2.1). If condition 3 holds, $S$ will decrease the utilization rate of link $l$ and resume the observation period. Within this period, $S$ will determine if the reduced rate does not destabilize its queue, and if the objecting active source is no longer bothered by the new session.

The description of the setup phase given above is somewhat simplified. In a realistic implementation of the protocol, one of the sources would have to initiate the session by sending a "connection request". Then, it would wait for the other party to respond. Only then, would the pair of hosts start exchanging dummy packets effectively beginning the phase described above.

This initial stage of the setup phase does not affect the ability of the protocol to provide a long term quality of service in a deflection network (although it does affect the length of the setup phase). For this reason, in our simulations, the setup phase begins as described above, that is, assuming that both parties already know about each other.

### 3.2.1 Blocking a connection

A session is blocked if *any* of the following events has occured during the last observation period of $F_Q$ frames: (i) the packet queue at $S$ has grown; (ii) $S$ has received an SD request from some source and it cannot slow down according to formula 3; (iii) $S$ has received a connection reject

(CR) message from the receiver at $\hat{S}$ (Section 3.3); or (iv) $S$ has received a connection terminate (CT) message from the corresponding source $\hat{S}$.

In the first three cases, $S$ will send a CT message to the peer source $\hat{S}$, to notify it that there is no point in continuing the setup phase. Note that, in order to speed up this process, CT might be sent with a high priority insertion.

#### 3.2.2 Accepting a connection

If source $S$ in the setup phase has not been blocked during the last observation period of $F_Q$ frames, it will check if a connection confirm (CC) message has arrived from the peer receiver. If it has not, $S$ will resume the observation period—a source cannot begin the active phase without making sure that the QoS perceived by the receiver is adequate.

As soon as CC is received, $S$ will send a *connection accept* (CA) message to its peer. This way, $\hat{S}$ will know that as far as $S$ is concerned, the QoS is likely to be sustained. However, this does not yet mean that the session is accepted—$S$ still does not know if the backward transmission from its peer source $\hat{S}$ can be sustained, and if the QoS perceived by the receiver at $S$ is acceptable.

A session is finally admitted if $S$ receives both CA and CC from $\hat{S}$, and $\hat{S}$ receives CA and CC from $S$. At this point, the transmission of "regular" packets begins and the session may not be blocked any more. We say that the session enters the *active phase* (see Section 3.1).

Having sent a CA message to its peer, the source slightly changes its behaviour. Namely, it can no longer be blocked by other (active) sources in the network. This is achieved by changing the setup field in the headers of the packets sent from $S$. Active sources no longer perceive these packets as coming from a setup source. Thus, $S$ behaves like an active source but it may still be blocked by its peer.

### 3.3. The receiver

We assume that the quality of service is described by the maximum packet loss rate $P_{max}$ and the maximum end-to-end delay $T_{max}$. Similar to the sender, the receiver may be in the setup or active phase. The initial size of the playout buffer, whose purpose is to reassemble possibly misordered packets and smooth out the jitter, is minimal and equals $B = 2N_F$ packets, where $N_F$ is the frame duration, i.e., the (average) number of packets in a frame.

A receiver in the setup phase determines if the perceived QoS measures do not exceed $f * P_{max}$ and $f * T_{max}$, where $0 < f < 1$. Fraction $f$, set in our simulation experiments to $0.5$, allows the receiver to respond to possible violations of the QoS before they actually occur.

Every $F_Q$ received frames (Section 3.1), the receiver checks if the delay suffered by the received packets is greater than $f * T_{max}$. If it is, the session has to be blocked—to decrease the delay, the receiver would have to decrease the size of the playout buffer to less than 2 frames. Then, the receiver sends a *connection reject* (CR) message to the sender and ignores the remaining packets arriving from the peer.

If the maximum delay is not exceeded, the receiver checks if the packet loss suffered over the last $F_Q$ frames exceeds $f * P_{max}$. If it does not, the receiver will begin a new observation period of $F_Q$ received frames. If, after some time [12], the packet loss and delay do not exceed their maximum values, the receiver will send a *connection confirm* (CC) message to the source, notifying it that the QoS at its end is acceptable. The receiver will begin the active phase in which it will only check whether the packet loss does not exceed $f * P_{max}$, as explained below.

If the packet loss exceeds $f * P_{max}$, the receiver checks if the current packet loss is greater than the packet loss recorded previously (i.e., from the previous observation of $F_Q$ frames), which would mean that the packet loss increases. If this is the case, the receiver increases the size of the playout buffer by one frame (i.e., $N_F$ packets), and refrains from removing the next frame from the buffer. This will increase the effective size of the playout buffer. Since packets will stay longer in the buffer before they are removed and played, fewer packets are likely to arrive too late and be dropped in the future.

If the current packet loss exceeding $f * P_{max}$ is less than the packet loss recorded previously, the receiver will not increase the size of the buffer. This is because the observed packet loss may exceed $f * P_{max}$ for some time despite the fact that no packets are being dropped any more. Obviously, increasing the playout buffer size would be pointless in such a case. In other words, the size of the playout buffer only increases if the current packet loss exceeds $f * P_{max}$ and if it is greater than the previous recorded value of packet loss.

The guaranteed packet loss $P_{max}$ may be *temporarily* exceeded. Detecting a packet loss exceeding $P_{max}$, our protocol will immediately increase the size of the playout buffer. If this buffer appears to be sufficiently large, packets will no longer be dropped. If they continue to be dropped, the buffer size will further increase until the required packet loss is achieved. In this context, $P_{max}$ can be understood as a "soft" guarantee—once a loss violation occurs, the protocol responds appropriately so in the *long term*, the packet loss is kept below $P_{max}$.

High load in the network makes arrivals of packets at the receiver less regular, which in turn increases packet loss. If the loss exceeds $f * P_{max}$, the buffer size grows, increasing the end-to-end delay. At some point, if the load in the network and the packet loss perceived by the receiver are sufficiently high, the size of the playout buffer may increase such that the end-to-end delay exceeds $T_{max}$.

However, our protocol does not allow the load to overly increase because all sources that inhibit already active sources are blocked. The network load stabilizes at some point, so that the packet loss, and thereby the end-to-end delay, cannot grow beyond $T_{max}$. Moreover, the observation time should allow the sources to detect possible delay violations during the setup phase. Such violations will result in blocking the session.

## 4. Simulation results

Synthetic fractal (self-similar) traces used in the simulator were obtained from the traffic generator programmed on the basis of the algorithm given in [16]. We adjusted the load, the Hurst parameter and the number $M$ of FRPs so that the synthetic trace obtained from the algorithm was close to the actual trace of a videophone. The simulation parameters were as follows: (i) network size $N = 100$ switches; (ii) the fraction $R$ of the playout buffer was set to $0.8$, that is, the size of the active part of the playout buffer was $B_R = 0.8 * B$[3]; (iii) packet size was $53$ bytes (corresponding to an ATM cell); (iv) all links were of the same length equal to 1 packet (i.e., 424 bits); (v) every source in the setup or active phase generated $24$ frames per second; (vi) the initial number of active sources in the network was $20$; (vii) every $20$ seconds, $P_c = 4$ (i.e., 8% of the network) of randomly selected pairs began their sessions.

Our model of a videophone session was borrowed from [2], which presents the trace of a videoconference session. We assumed that the trace of a videophone session would be similar—the only difference being that in a videophone session, every user transmits/receives video to/from a number of other users.

On the basis of this trace, we set the average load $\lambda$ of the traffic between a pair of active switches to $10$ kB per frame. Assuming the rate of $24$ frames per second, we obtain the average load of $1.97$ Mb/s computed from a sample of $1000$ frames.

The Hurst parameter of the traffic was uniformly distributed between $H_{min} = 0.60$ and $H_{max} = 0.75$, and the number $M$ of FRPs was set to $15$. The trace produced with these values of $H$ and $M$ corresponds best to the trace given in [2].

We have performed experiments for 2- and 4-connected networks. In the 2-connected case, the network capacity $C$ is $10$ Mb/s. Assuming that it takes a signal $5 * 10^{-9}$ sec. to travel 1 meter in the medium, the physical distance between a pair of neighbouring switches is about $8.5$ km.

We monitored the overall throughput, as well as the packet loss, and end-to-end delay of individual sessions vs. the number of active sources. The following three environments were considered:

**E1** No QoS enforcement mechanism. There is no control over the quality of service and the only limit imposed on the number of active pairs in the network is the network size.

**E2** With a variant of the proposed control mechanism in which the receiver cannot adjust the size of its playout buffer or affect the decision about the admission of a new session to the network.

**E3** The full control mechanism as described in Section 3.

In environments $E2$ and $E3$, we also measured how many initiated sessions were blocked in the setup phase due to the high network load. The initial size of the playout buffer was $2$ frames. Note that this size cannot change in environment $E2$. The sequence of peers attempting to start up their sessions was the same in all three environments.

If we measured the packet loss by simply adding all dropped packets and dividing this number by the number of all received packets, its value would be misleading. Even the loss of a few packets could make the value of this performance measure very large for the duration of an entire session. For this reason, packet loss was reset every $20$ seconds immediately before a new set of $P_c$ pairs of sources begun their setup phases. This makes it easier to compare various environments, and it shows how adjusting the playout buffer size helps the receiver sustain the long term quality of service.

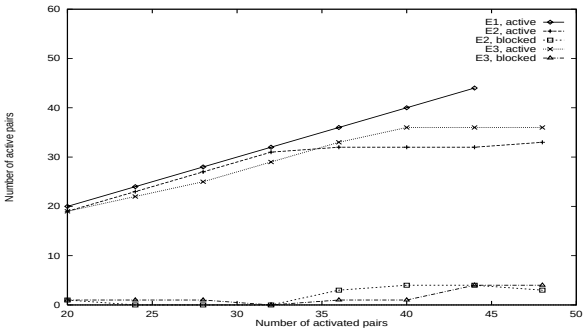
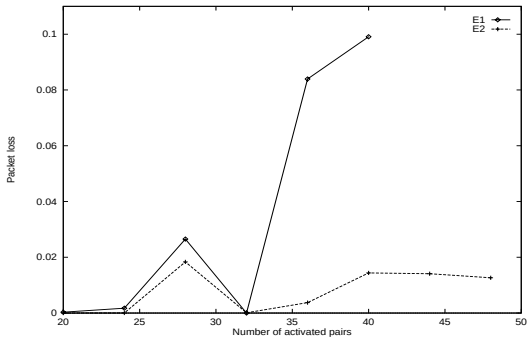


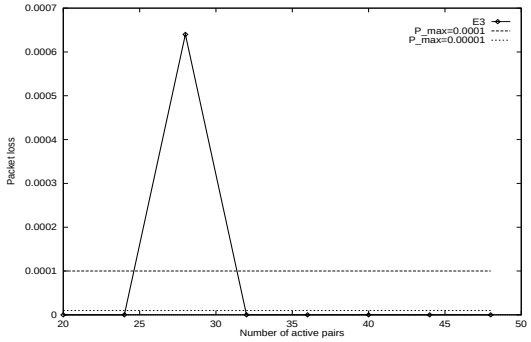

**Figure 2. Number of blocked and active pairs vs number of activated pairs.**

Figure 2 shows how many new pairs of sources are blocked and admitted to the network (based on the averaged results from $10$ experiments). The number of randomly selected pairs that every $20$ seconds try to begin new sessions is $P_c = 4$. The number of activated pairs in the network grows linearly with the number of pairs that try to begin

[3] Our experiments reported in [15] indicate that the best value of $R$ (for all deflection networks and traffic patterns) is about $0.8$.




(a) Packet loss, E1, E2




(b) Packet loss, E3

**Figure 3. Packet loss vs number of pairs**

their sessions only in $E1$. The number of blocked sessions in $E1$, not shown in the figure, is of course 0.

It is not the case in the other two environments. When the number of pairs in the network exceeds a certain threshold, new sessions are blocked. That is why, the number of active sources eventually becomes constant, and the number of blocked sources becomes $P_c$ (which was set to 4).

Figure 3 shows a sample history of packet loss at one of the receivers that suffered some loss almost from the very beginning of the session. As we have mentioned above, packet loss (and also delay) are reset every 20 seconds, so these two performance measures show the situation in the network within such intervals. As before, the number of randomly selected pairs that every 20 seconds try to begin new sessions is $P_c = 4$.

In environment $E1$ (Figure 3(a)), packet loss generally increases with the addition of every $P_c$ new pairs. The only exception is for 32 active pairs — apparently, the activity of other sources in the network did not increase the jitter perceived by the receiver to the point in which packets had to be dropped.

Packet loss in environment $E2$ is much lower because fewer sessions are admitted to the network. Limiting the number of active sources limits the load, jitter and thereby packet loss.

Figure 3(b) shows the packet loss at the same receivers as well as the required guarantee imposed on packet loss in environment $E3$. At certain point, the packet loss exceeds $P_{max}$. In that period of 20 seconds (or 480 frames) 60 packets from one frame were dropped. When the protocol at the receiver side detected this, it increased the playout buffer by 1 frame which was sufficient to sustain zero packet loss in the remaining part of the session.

As we mentioned in Section 3.3, packet loss may temporarily exceed the allowed maximum value. However, after the receiver adjusts its playout buffer, the required quality of service is achieved.

End-to-end delay and throughput (not shown here) are consistent with our expectations. The delay perceived at the selected receiver in environment $E1$ increases with the increasing load because of a longer queueing time at the sender and increased propagation delay caused by deflections. In environment $E2$, in which some sources are blocked, the end-to-end delay is much lower because the load in the network does not grow beyond a certain threshold. In environment $E3$ the delay is most regular.

The throughput in environment $E1$ increases with the increasing load until the network becomes saturated. As we could expect, in the other two environments the throughput is smaller (by about 20% for the highest load) which is caused by the smaller number of active sources in those networks.

## 5. Possible improvements of the protocol

It may make sense to keep *signatures* of typical sessions, i.e., pre-generated sequences of frames to be used for session probing during the setup phase. Such signatures could be viewed as implicit QoS specifications. In our simulations, we set the initial size of the playout buffer to $2N_f$ packets. In fact, the receiver could start its setup phase with a buffer size greater than 2 frames. This would not only result in a shorter setup time but could also ensure that no packet would be dropped during the session. Note that in the experiment shown in Figure 3(b), the buffer of size 3 frames gives zero packet loss throughout the session. See [12] for more on the setup time and the estimation of the initial size of the playout buffer.

To decrease the protocol's complexity, a setup source that receives a slow-down message SD may be simply blocked—it cannot slow-down. This way, the setup phase will become shorter, and active sources will not have to distinguish between packets sent from the setup source via different outgoing links. The only downside is about 3% de-

crease of throughput. Since every SD message received by a setup source forces it to block the session, fewer pairs will be allowed to initiate their sessions.

## 6. Summary and future work

Our results suggest that limiting the number of active sources coupled with the receivers' ability to adjust the size of their playout buffers allows the network to provide a long term quality of service. Notably, no resources are reserved in advance and no bandwidth is wasted on tokens, although some resources are used in the setup phase. This also suggests that the proposed protocol is easily scalable. In fact, with the increasing size and/or capacity of the network, only the setup phase needs to be extended.

Applications other than the videophone would require some minor changes to the protocol. For example, in the setup phase of a videoconference, the connection accept (CA) message would be multicast. Every source has to wait for CA from all other sources before beginning the active phase. The arrival of at least one connection terminate CT message, which would often be multicast as well, would block the session (or perhaps trigger a new connection attempt with reduced QoS requirements).

If we review the properties of our protocol against the list of properties expected from the ideal protocol [6], we shall see that it has several advantages, including simplicity, predictability [10], the capacity-1 property, zero access delay under light load, and the ability to sustain synchronous traffic up to using the whole bandwidth of the network. Notably, it allows to sustain a desired quality of service without explicit reservation of resources at intermediate nodes.

In [14] we have proposed a simple scheme that improves the performance of a stream traffic regardless of the intensity of datagram traffic. In this paper, we have considered only stream sessions. It would be interesting to merge these two schemes and investigate the performance of a network under mixed (datagram and stream) traffic scenarios. We believe that the performance of our protocol, and thereby a stream traffic, would not suffer even if it shared the network with bursty datagram traffic. As we showed in [15], deflection networks flexibly handle bursty traffic sources, and the protocol presented in [14] allows to throttle excessive datagram traffic.

## 7. Acknowledgements

One of the authors* would like to thank Mustapha Aissaoui, Ron Huberman and Gerald Bloch from Alcatel Networks Corporation for their support. The authors also thank anonymous reviewers for their useful comments.

## References

[1] C. Baransel, W. Dobosiewicz, and P. Gburzynski. Routing in multi-hop switching networks: Gbps challenge. *IEEE Network Magazine*, (3):38–61, 1995.

[2] J. Beran, R. Sherman, M. Taqqu, and W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 3:1566–1579, 1995.

[3] F. Borgonovo and E. Cadorin. Locally-optimal deflection routing in the bidirectional Manhattan network. In *Proceedings of IEEE INFOCOM'90*, pages 458–464, 1990.

[4] F. Borgonovo and L. Fratta. Deflection networks: architectures for metropolitan and wide area networks. *Computer Networks and ISDN Systems*, (24):171–183, 1992.

[5] A. Choudhury and N. Maxemchuk. Effect of a finite reassembly buffer on the performance of deflection routing. *Conference Record of the International Conference on Communications (ICC)*, 3, 1991.

[6] W. Dobosiewicz and P. Gburzyński. On token protocols for high–speed multiple–ring networks. In *Proceedings of the International Conference on Network Protocols*, pages 300–307, San Francisco, USA, Oct. 1993.

[7] M. Garret and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 269–280, London, UK, Sept. 1994.

[8] M. A. Marsan, E. Leonardi, F. Neri, and G. Pistritto. Service guarantees in deflection networks. In *6th IEEE Workshop on Local and Metropolitan Area Networks*, San Diego, California, Oct. 1993.

[9] N. Maxemchuk. The Manhattan Street Network. In *Proceedings of GLOBECOM'85*, pages 255–261, 1985.

[10] N. Maxemchuk. Routing in the Manhattan Street Network. *IEEE Transactions on Communications*, 35(5):503–512, May 1987.

[11] N. Maxemchuk. Problems arising from deflection routing. In Pugolle, editor, *High Capacity Local and Metropolitan Networks*, pages 209–233. Springer Verlag, 1991.

[12] W. Olesinski. *Connection-less Paradigm in High-Speed Networking*. PhD thesis, University of Alberta, June 1999.

[13] W. Olesinski and P. Gburzynski. Multicast in deflection networks. In *Proceedings of Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, pages 50–55, Montreal, Canada, July 1998.

[14] W. Olesinski and P. Gburzynski. Quality of service in deflection networks. In *Proceedings of 9th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'98)*, pages 11–16, Banff, Canada, May 1998.

[15] W. Olesinski and P. Gburzynski. Real-time traffic in deflection networks. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)*, pages 23–28, San Diego, California, Jan. 1998.

[16] B. Ryu and M. Nandikesan. Real-time generation of fractal ATM traffic: model, algorithm, and implementation. Technical report 440-96-06, Department of Electrical Engineering and Center for Telecommunication Research, Columbia University, New York, NY, mar 1996.