Renegotiable Quality of Service – A New Scheme for Fault Tolerance in Wireless Networks

Tsu-Wei Chen, Paul Krzyzanowski, Michael R. Lyu, Cormac Sreenan and John A. Trotter
Bell Laboratories
Lucent Technologies, Murray Hill, NJ 07974

Abstract

In this paper, we propose the concept that faults in telecommunications networks often manifest themselves as reductions in service quality, which can be addressed by using the notion of Quality of Service (QoS). In wireless ATM networks, the ability to provide QoS guarantees for high priority traffic in the presence of noise or faults is of utmost importance. Moreover, there is a need for renegotiating existing QoS on an established connection, since the characteristics of a wireless link may well change during the lifetime of a connection due to mobile hosts' movements or external interference. In this paper we describe a general QoS strategy as a fault tolerance mechanism, and address the problems associated with providing QoS over a wireless link. We present a QoS scheme with renegotiation capability, define an API (application programming interface) for the access to this scheme and describe our implementation for this QoS API on the SWAN system, a wireless ATM network, and summarize its performance using measurements obtained from a series of experiments based on different fault scenarios.

1 Introduction

Conventional fault tolerance schemes in treating computer or network failures rely on the design and allocation of protective redundancies [10]. In typical communication networks, the protective redundancies can be applied in multiple levels of system components, including processors, memory units, disks, communication links, data streams, transmission time frames, and software codes [6]. For example, a number of redundant disk array architectures (RAID) are proposed in [8], on which a clustered architecture can be built [7]. On-line failure recovery algorithms can allow a fast recovery process by either absorbing the disk bandwidth not consumed by the user processes [5], or by utilizing the inherent redundancy in video streams of the application [14].

We consider fault tolerance in a system of communicating processors that work together to provide some service. An example of such a system might be a cluster of workstations providing web service to a client. The client requests a page and the server machines respond by delivering that page. A further example is a video delivery system where a server delivers a video stream to a client connected to the network. In both scenarios a fault might be a service interruption due to network component failures or machine crashes.

Traditional fault tolerance has studied the problem of dealing with service outages because components fail in a system. Techniques involve replacing the faulty element, either using a hot standby or a warm standby or even accepting some downtime and repairing the unit off line. Existing traffic may also be rerouted via another redundant path from the source to its destination by routing algorithms [2, 12].

In telecommunications networks, fault tolerance has progressed to considering the recovery of failures such as reductions in the speed or capacity of a service. For instance, a telecom switch might reduce the maximum number of calls that can be handled if a processor failure is encountered. Thus faults, rather than manifesting themselves as complete service failures, often manifest themselves as reductions in service quality such as reduced bandwidth or increased latency, e.g., due to rerouting. These problems can be addressed in the context of communications networks by using the notion of Quality of Service (QoS) [3].

We view QoS as a new scheme in providing fault tolerance to prevent service interruptions in a proactive fashion: In the presence of faults, services that need a level of resource (e.g., bandwidth) will be guaranteed to perform satisfactorily when there is enough of the resource to be shared, and when it is scarce, negotiation with the service provider will take place to assure a lower, but deliverable level of service.

Wireless networks pose a more demanding set of challenges than in wired networks. Due to movement of mobiles, a fault-tolerant protocol for maintaining location directories in mobile networks is needed [9]. Furthermore, low signal-to-noise (SNR) ratio makes wireless link errors a norm rather than an exception in the system. The data rate does not drop immediately to zero; instead, it slowly reduces as noise increases. Thus before completely switching to a redundant channel on a faulty link, the link errors can be gracefully tolerated by the provisioning and renegotiation of QoS in a (high) service level, in addition to existing recovery techniques using error correcting codes or retransmission in a (low) physical level.

In this paper we propose an extension to the QoS concept of "renegotiation" which can be used to reestablish a level of reliability for a system. We explain how such a system works using a wireless network data link as an example. The approach allows applications to specify a desired quality over the link. If a failure occurs and the available bandwidth decreases, the system renegotiates the bandwidth requirements. In this way the application is knowledgeable about what level of service is available and may be able to change what it does accordingly.

This concept allows a system to tolerate failures in parts, and makes it a system for recovering in a graceful way transparent to its users. The remainder of this paper is organized as follows: In Section 2 we discuss a general QoS approach and its constraints, while in Section 3 we describe a wireless ATM network environment used as a testbed for our QoS design, implementation, and experimentation. Section 4 lays out the architectural policy and mechanism for our QoS scheme, and presents the implementation in detail. In Section 5 a number of fault tolerant experiments are conducted to verify our scheme and compare it with a traditional approach without QoS. Conclusions are drawn in Section 6.

2 The QoS Approach

The need to make computer communication have more deterministic behavior is the origin of work on QoS. Traditional packet networks provide a single service model that makes a best-effort attempt to deliver data packets. Bandwidth is shared out amongst competing senders on an as-needed basis. Packets are not guaranteed to take the same route or experience the same delay in getting to their destination. This is not important for traditional computer applications, as long as the overall delays are not excessive. On the other hand, in circuit switched networks, the service model is highly predictable, with a fixed slot of bandwidth allocated for use by a sender in each time period, and with equal delivery times for each slot. This has proved useful for transferring digitized voice at the fixed rate of 64 Kbps. The desire to operate just a single network for both computer and telephony traffic sparked work on integrated services networks. In addition to these service categories, such a network should also support multimedia services in the form of packet audio/video, and real-time services like process control which have strict communication delay constraints. Multimedia services typically demand high bandwidths and are sensitive to delay and variation in delay, but may be prepared to tolerate some data loss. For example, dropping one image from a video sequence at 30 images/second may not be noticeable. Real-time applications usually have low bandwidth requirements, but demand predictable delay and zero loss. Support for such diverse requirements is expressed in terms of the particular QoS expected from the network.

The challenge of designing a network to address these issues led to the development of asynchronous transfer mode (ATM), ATM transports data in small, fixed size packets called *cells*. Small cells have the benefit of increasing scheduling granularity and hence providing more control over queueing delays. This avoids problems such as a delay sensitive audio packet getting delayed behind a large file transfer packet. Having a fixed cell size allows the network design to be more deterministic. ATM carries cells across the network on connections known as virtual circuits (VC). In essence, a VC is just a way of maintaining state for a particular flow of data at each stage in its path from source to destination. A key element of this state information relates to how the cells for a VC are processed in order to satisfy its QoS requirements. Hence, in ATM the concept of setting up a VC with an associated QoS exists. Setting up a VC involves taking information on the traffic and expected performance and negotiating along a path in order to reserve the necessary resources, such as switch buffer space. Performance information describes any requirements on delay and delay variation for cells in a VC. Using this information the network checks to see if the necessary resources are available. If they are, then the VC is set up, otherwise the request is denied. This process is known as admission control. Once admitted, the network continually checks that the VC sends data according to its allowance, known as *policing*. It also schedules cells at the switches in order to achieve the agreed QoS.

Our work is motivated by the rising popularity of wireless data networking and the desire for fault tolerant communications. Wireless networking is inherently unreliable. Various forms of interference on the wireless link result in changing bandwidth availability and low effective bandwidths due to high error rates. These problems are exacerbated as users move around. Faults of this kind require a fresh look at how such networks can be used to support applications which demand some degree of predictability. We adopt the approach of ATM, in which QoS is used to form a service contract between applications and the network. We build on that work by recognizing that an unreliable wireless network demands a more dynamic approach to resource usage. Many applications can deal with varying bandwidth availability once provided with sufficient knowledge of the resource climate. Typical examples include audio and video applications which can alter their rate or encoding to match the available bandwidth or deal with different error rates. Our contribution is a QoS scheme which builds on this notion of adaptation by providing explicit renegotiation. This is similar in spirit to the feedback mechanisms for non real-time traffic in ATM, but differs in that we aim to provide feedback right up to the application level, not just to the sending host [11]. Thus we incorporate renegotiation as a key part of our QoS API.

Four elements compose our approach. First, we engage the support for multiple VCs over a wireless channel, and the usage of a set of per-VC QoS parameters to influence bandwidth allocation. Second, we define a group of interface routines for opening, accessing and closing VCs, as well as being able to assign QoS parameters. These parameters include a description of the traffic type as constant bit-rate (CBR), variable bit-rate (VBR) or available bit-rate (ABR). CBR and VBR applications have real-time requirements, e.g. 64 Kbps speech and compressed video. ABR is used for more traditional applications which can accept much more variability in service. Consistent with our emphasis on adaptation, we accept values for preferred and minimum bandwidths for a VC. Third, we design a centralized QoS manager to coordinate the access to a wireless channel. Using parameters supplied via the API, the manager performs admission control, monitors performance on the channel and initiates renegotiation when necessary. Fourth, for each VC, the application provides a callback routine which is used by the QoS manager to provide feedback as part of renegotiation.

The QoS scheme described above is successfully implemented in a wireless ATM network: SWAN [1].

3 The SWAN Environment

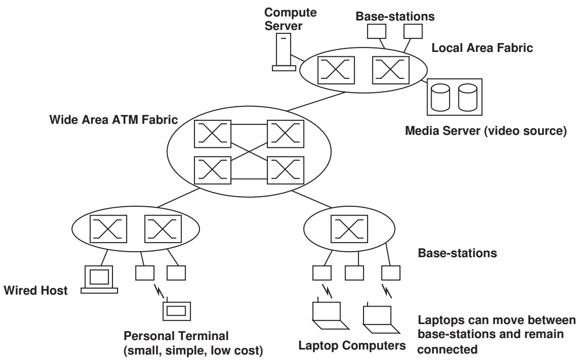

Fig. 1: SWAN system architecture

The SWAN (Seamless Wireless ATM Network) system, shown in Fig. 1, is a testbed for wireless networked computing. SWAN consists of mobile units which are usually laptops, and base stations which are connected to a backbone network [4]. Both the base stations and laptops are equipped with a radio interface known as the FAWN (Flexible Adapter for Wireless Networking) [13] card that allows them to communicate with each other wirelessly. Each base station has a range of 100 feet inside a building, providing access to a local area network for mobiles in their vicinity. As well as communicating with the base stations the mobiles can communicate with each other, allowing them to create ad-hoc networks that continually change as the mobiles move around.

The FAWN card provides a very programmable platform on which to develop interface software, which is important in a testbed. FAWN uses a 2.4 GHz ISM band radio modem whose raw bit rate is 624 Kbps which is divided between incoming and outgoing connections. The modem has a raw error rate of 1×10^{-5} for a signal strength of -77 dBm which translates to a packet loss of one in 1500 for our 64 byte packets. The FAWN adapter has four 64 byte packet buffers implemented in hardware to store buffer complete packets and therefore improve performance. The FAWN card is controlled by an ARM610 processor which takes the packets from the buffers, processes them and makes them available to a host computer via a PCMCIA interface.

A simplified diagram of SWAN's channel access scheme is shown in Fig. 2. A TDD (Time Division Duplex) scheme is used to share the bandwidth between the base station and the mobile host. The traffic of each direction alternatively transmits a data burst of 10 ATM cells at a time and then switches to receiving mode for data from the other direction. Due to the overhead introduced by the TDD scheme and the ATM cell structure used in SWAN, the available bandwidth is 240 Kbps in each direction.

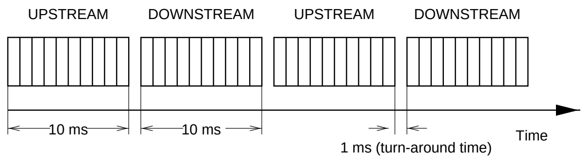

Fig. 2: TDD scheme of SWAN

Several other wireless communication devices are available for the local area network market (e.g. WaveLAN, RangeLAN), and most of them are based on or akin to the IEEE 802.11 or CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) schemes. Though a CSMA/CA scheme simplifies the hardware implementation and provides reasonable efficiency in supporting datagram traffic, its random access characteristic cannot provide multimedia applications with a predictable bandwidth available on the link. The SWAN system, as described previously, was designed with ATM traffic in mind. Its TDD scheme assures a constant bandwidth can be granted, at least when the channel condition is stable. Therefore, SWAN provides a good platform for realizing our proposed QoS scheme.

Unfortunately, despite the direct support of ATM cells and TDD MAC scheme, challenges still exist in SWAN when considering the support of multimedia traffic in a wireless, mobile environment. These challenges include (1) low radio bandwidth; (2) increasing likelihood of erroneous packets due to lower SNR; (3) lack of a mechanism to support traffic of different classes and (4) lack of an interface to provide link QoS information to upper layer applications. We will address our approach to these problems when describing our QoS implementation in the next section.

4 System Implementation

To achieve QoS renegotiation in a unreliable wireless environment, We design a mechanism that exists in the operating system level (the interface at which the applications request resources) for an application to request a grade of service for a network connection and for the application to be informed about changes on that connection. In this section, we describe our approach in detail from two aspects: (1) the policy, in which we determine how an application can specify its QoS requirements and how it can be notified of failures so it may adapt to a new environment; (2) the mechanism, where we addresse the realization of the policy.

4.1 Policy

Our goal is to use existing interfaces and facilities provided by widely accepted operating systems, instead of creating an ad-hoc system or proposing a new, proprietary interface. Therefore we chose Linux, a UNIX-like system, to develop our work. Also, since SWAN is designed to provide ATM connectivity, we consider the QoS negotiation on a per-VC basis. In our approach, the VCs are instantiated as UNIX devices, such that one may use the `open()` system call to obtain a VC and the `close()` system call to release the VC. During the connection, data is sent and received via `write()` and `read()` system calls.

As soon as a circuit is activated (opened), it is given a default service grade of unspecified bit rate (UBR) service. This allows applications that do not have QoS demands to receive the best service effort from the system. If an application does wish to specify its bandwidth need, it does so with one or more `ioctl()` system calls (I/O control).

By performing these `ioctl()` operations, an application may select ABR, CBR, or UBR services, and specify the associated QoS parameters. Currently, two bandwidth parameters (minimum and preferred) have been considered. Supporting these two bandwidth parameters allows an application to specify a range of acceptable bandwidth so that it doesn't get informed each time when the supported bandwidth changes.

The way an application should be notified of QoS failure is also considered. Using existing UNIX facilities, the signal mechanism allows the operating system to send a "QoS failure" message to the application. The application uses the `signal()` system call to setup an exception handler to process this QoS failure event. A similar policy exists for the reverse operation, where an application receives a signal when the service failure is removed and returns to its original performance.

4.2 Mechanism

In this section we describe in detail the realization for the above policy. We first introduce the usage and functionality of this API, by which the applications specify the QoS parameters associated with the VC. Then we draw the core of the implementation.

4.2.1 The API The first aspect of creating the desired interface is to provide a device driver for the VCs and the associated API to manage them. The VCs are implemented as devices within the UNIX file systems to which the standard system calls can be applied. The entire API of our implementation is shown in Fig. 3. Since it is implemented using standard UNIX I/O operations, a user program can manipulate its connection just as an ordinary character device.

The QoS requests are made through the `ioctl()` system call with the application specifying parameters for the type of service, minimum or preferred bandwidth, etc. The parameters we have implemented for the QoS negotiation are listed in Table 1. The default values shown in the table indicate that an UBR service is assumed to reserve the system minimum bandwidth, which is zero, if the application does not make any QoS request.

QOS_REQUEST	ARGUMENT	DEFAULT
VC_SERVICE	ABR,CBR,UBR	UBR
VC_MIN_BW	n (Kbps)	0 (Kbps)
VC_PREF_BW	m (Kbps)	0 (Kbps)

Table 1: ioctl(): parameters

With this API, the application can be easily programmed using a traditional client/server model: (1) the client and the server first request a VC using `open()`. (2) If the VC can be opened successfully, the required QoS

```
int open(char *vc_dev_name, int mode);
      /*  acquire a VC, return -1 if requested VC is in use  */
int close(int vc_des);
      /* release a VC  */
int read(int vc_des, char *buff, int n);
      /*  read n bytes from a VC  */
int write(int vc_des, char *buff, int n);
      /* write n bytes to a VC  */
int ioctl(int vc_des, int qos_request, long arg);
      /* request, negotiate a QoS attribute of a VC  */
int signal(int QoS_SIGNAL, void *qos_handler(int));
      /*  set up a handler for QoS changes  */
```

Fig. 3: List of the API

can be provided by using `ioctl()` with parameter values based on the traffic characteristic. (3) Data is transmitted using `read()` and `write()`. (4) When the program terminates, VCs should be released by using `close()`. The system routine `signal()` listed in Fig. 3 is not really a QoS operation; instead it is used by the application to setup its own QoS interrupt handler which can renegotiate new QoS agreements when the original service requirements cannot be met.

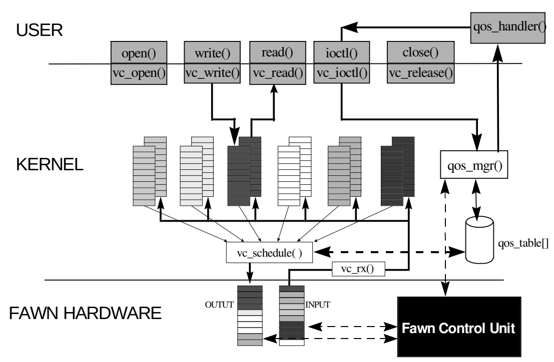

Fig. 4: The architecture for multiple queues scheme

4.2.2 Multiple Queues Scheme Fig. 4 illustrates the block diagram that shows the relationship between user applications and FAWN hardware, as well as the interaction among each module. On the top, applications in user level communicate with the QoS mechanism through a set of interface routines (the API). A group of priority queues are dynamically allocated in the kernel space. Each of these queues corresponds to an individual VC. Once a VC is opened and its QoS is negotiated through `ioctl()`, which interacts with the QoS manager (`qos_mgr()`) for service and bandwidth specification, the QoS manager translates the requested service type and bandwidth in terms of time slots for carrying data cells in each data burst. This information will be kept in a QoS table (`qos_table()`) that will later be referred by the VC scheduler (`vc_schedule()`). The QoS manager is also responsible for monitoring the overall link quality through the FAWN hardware, and providing feedback directly to application when the requested QoS can not be satisfied or when a better service is available. This feedback is implemented through the UNIX signal, as described in Section 4.2.1.

The VC scheduler reads packets from those activated queues and sends them to the FAWN hardware for transmission. It serves these multiple queues in a "round-robin" fashion which allows a control of QoS granularity such that one circuit will not dominate the data path with a large chunk of data.

5 System Functions

Currently, two major functions provided in this system are bandwidth reservation and QoS renegotiation. They are the keys to providing multimedia traffic support in wireless networks.

5.1 Bandwidth Reservation

In SWAN, the radio channel in use is shared between a base station and a mobile host in a TDD fashion. Thus the allocated bandwidth can be represented in terms of the number of time slots devoted to a connection. For example, a connection granted with one slot in each data burst is served at the bit rate of 24 Kbps (240 Kbps/ 10 slots) in FAWN's TDD scheme.

During bandwidth reservation, the QoS manager is responsible for converting the bandwidth requirements into the necessary number of time slots for transmitting the data to meet the bandwidth guarantees. If the time slots cannot be allocated, the bandwidth request will be rejected by the QoS manager. UBR service is provided by placing data in slots that are unreserved or unused by CBR/ABR circuits. In addition to providing service to queues of different QoS requirements, a starvation prevention scheme is also utilized to prevent starvation on UBR service. In this scheme, at least one data slot is reserved and shared among all UBR queues in a "round-robin" fashion so no UBR connection will be starved even if some of them are heavily loaded. This avoids any dominate usage of one application over the bandwidth of the wireless link.

5.2 QoS Renegotiation

In a wired network, QoS is usually guaranteed for the life time of each connection. In a wireless network with host mobility, however, such a guarantee is not realistic due to distance, noise or channel fading, etc. On the other hand, many multimedia applications have used algorithms that can adapt to bandwidths that users specify. For instance, several video transmission schemes (nv, vic, etc.) can adjust their resolution and frame rates to fit the bandwidth parameters that they are given; several audio applications may adjust their sampling rate and level of quantization based on the channel bandwidth. Such properties have not been utilized for a self-adjusting multimedia application because of the lack of QoS feedback from the traditional packet network like the Internet, or even some ATM networks.

In our work, the signaling mechanism we propose informs the applications of the changes in QoS. The applications can benefit from this mechanism by simply setting up interrupt handling routines so that when they are notified of a change in QoS, they can adjust their data transmission algorithm based on the current QoS information. This signaling mechanism was implemented in the QoS manager which has a direct access to the FAWN hardware to learn about the current status of radio link. For example, when the QoS manager detects the decrease in radio bandwidth, it first reduces the service to the UBR traffic; If such a reduction is not sufficient to guarantee the requested bandwidth for all ABR and CBR traffics, it then reduces the ABR/CBR service rate to its minimum requirement. Finally if the bandwidth is still not sufficient, the QoS manager will prorate the assigned bandwidth on all CBR and ABR connections and signal the corresponding handlers created by the applications to notify the change of QoS. Upon receiving the signal from the QoS manager, the handler in each application can decide whether to accept the newly assigned QoS, to terminate the connection, or to renegotiate a new QoS through the provided API.

6 Fault Tolerant Experiments and Analyses

We conducted two experimental studies to verify the implementation of our QoS scheme, and assess the effectiveness of this scheme as a fault tolerant mechanism in the presence network failures. These experiments are described as follows.

6.1 QoS Renegotiation Experiment

The first experiment studies the effect of signal to noise ratio on a wireless link in the SWAN system. As signal to noise ratio decreases the number of erroneous packets received increases, which maps to a decrease in available bit rate over the link. We plot the performance of the system as the bit rate decreases (in other words as the error rate increases) by measuring the traffic through the system for a system based on UDP datagram transmission (non QoS system) as well as our QoS based system.

In the experiment we assume that there are three data streams, A, B and C sharing the wireless link from a mobile to a base station that is connected to the network. Stream A is an ABR stream like an *ftp* file transfer which can use as much data rate as possible up to some maximum. In our experiment this maximum was 48 Kbps. Streams B and C are CBR streams, like those used in an uncompressed video transmission. Stream B needs 72 Kbps and stream C 96 Kbps. Stream C can operate at the lower bit rate of 48 Kbps if it is informed of the change. It can achieve this by reducing the number of frames per second that it sends.

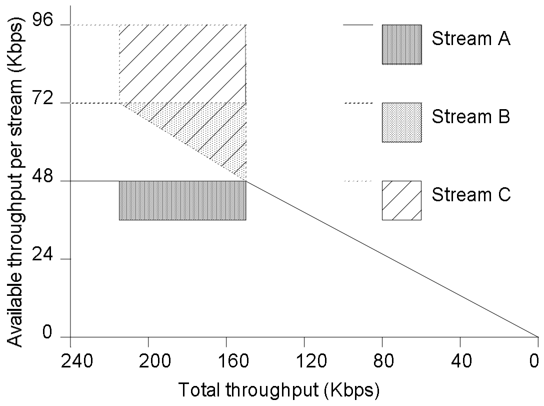

Fig. 5: Realizable throughput for a system without QoS

6.1.1 Variation of Throughput for the Non-QoS Case

The graph in Fig. 5 shows the variation of actual throughput versus the available data throughput for each of the streams A, B and C. The system's available data throughput varies along the x axis from 240 Kbps down to zero. The shaded regions in the graph indicate a range of possible bitrates that data streams achieve, and the actual data rate tends to oscillate between the maximum and minimum values in each region. The three streams, A, B and C are presented to the communication channel. In the first region from 240 to 216 Kbps of throughput, streams A, B and C (whose total requirement is 216 Kbps) are accommodated.

In the second region, from 216 through about 150 Kbps, the total bandwidth requirements of all the channels cannot be satisfied, and they begin to interfere with each other. The scheduler attempts to give one third of the total data rate to each of the channels. Streams B and C can consume the third that they are given. However, stream A under-utilizes the available data rate because it only needs 48 Kbps, while a third of the bandwidth in this region varies from 72 to 96 Kbps. This means that there is spare throughput that streams B and C attempt to use. Both streams have the potential of getting their full bandwidth at some instances of time, thus the minimum bandwidth in this region is set by a third of the available data rate and the maximum bandwidth by the maximum data rate that can be sent by each stream. Even though stream A's data rate requirements do not exceed one third of the available data rate, it is interfered with by streams B and C which are using some of that capacity by putting large packets in the single queue which delay stream A's packets.

The final range is from 144 through 0 Kbps. Here all three streams can consume a third of the available bandwidth and compete for the bandwidth evenly.

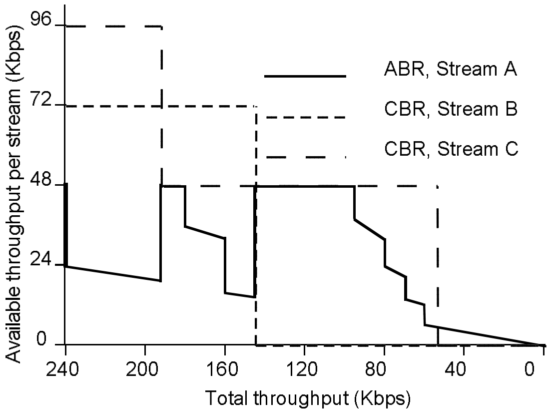

Fig. 6: Realizable throughput for a system with QoS

6.1.2 QoS Version In our implementation of the QoS scheme, all the queues are sharing the available bandwidth of 240 Kbps (with no errors). As the number of errors increases over the link the effective data rate for each of the queues decreases proportionately. Because the QoS scheme allocates bandwidth on a slot basis the granularity of the available data rate is one tenth of the total available bandwidth.

The graph in Fig. 6 shows how each of the streams A, B and C respond to variations in the available bit rate. When there are no errors on the link stream A can operate at 48 Kbps (consuming two of the ten available timeslots), stream B operates at 72 Kbps and stream C at 96 Kbps. As the error rate increases slightly both streams B and C need extra timeslots to continue to be provided with their required bit rate. One slot comes from slack in the system (only 9 of the 10 were in use initially) and the other comes from stream A, which is an ABR stream and is downgraded to 1 timeslot. As the data rate reduces further, eventually stream C is unable to have its requested 96 Kbps. At this point the QoS manager sends it a signal telling it to renegotiate its required bit rate, and since it can operate at 48 Kbps it does so. At about 50 Kbps stream C needs 3 timeslots to provide the 48 Kbps. This allows stream A to use another two timeslots for its ABR traffic. As the error rate increases the two CBR streams consume more timeslots, and correspondingly the bandwidth available for stream A reduces. At an error rate of about 95 Kbps stream B needs another timeslot to satisfy its data rate needs. However, since stream A always needs at least 1 timeslot and the CBR traffic of stream B cannot support a lower bit rate, it is renegotiated to zero. This makes more bandwidth available for stream A, but as the error rate further increases it gives that bandwidth to the CBR stream C which eventually stops when the error rate rises to about 185 Kbps, when the throughput falls to 55 Kbps.

6.2 Network Link Failure Experiment

The objective of the next experiment is to examine whether our QoS VC architecture and implementation scheme truly provides the required fault tolerant mechanisms in delivering the service it guarantees, and to compare the results with a network without QoS assurance. In this experiment, we consider link faults due to the failure of radio interface, particularly when the mobile stations move out of the radio range. That is, the quality on a failed link will degrade to a level where no data can be transmitted, and therefore traffic needs to be rerouted to another link in order to maintain session continuity.

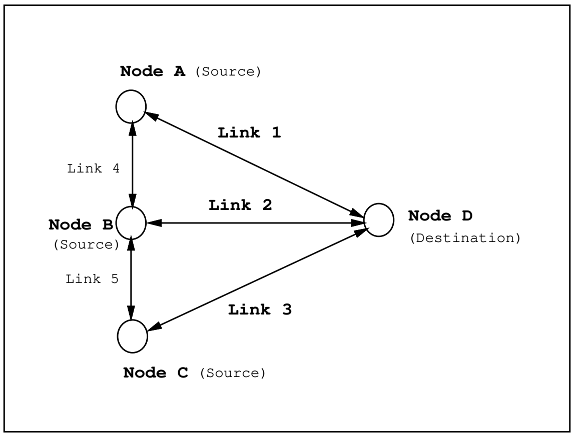

Fig. 7: Topology for a network link failure experiment

6.2.1 Network Topology and Traffic Flow Assumptions

We consider a multi-hop wireless topology based on SWAN environment as shown in Fig. 7. There are four nodes in this network topology: Node A, Node B, Node C, and Node D. Five SWAN radio links are set up for communication between these nodes, marked as Link 1 through Link 5 in Fig. 7. Without losing generality, we made the following assumptions during the our experiment:

1. Nodes A, B, and C are source nodes while node D is a destination node.
2. Initially, Node A sends a CBR traffic (CBR video 1, abbreviated as *v1*) to Node D via Link 1, Node B sends an ABR traffic (ABR datagram, abbreviated as *d1*) to Node D via Link 2, and Node C sends a CBR traffic (CBR video 2, abbreviated as *v2*) to Node D via Link 3.
3. Both *v1* and *v2* are uncompressed video sessions transmitted at frame rate of 0.5 frame/sec and 1 frame/sec, which yield the constant bit rate of 73 Kbps and 145 Kbps, respectively. The traffic *d1* is designed to represent the ordinary datagram traffic thus we assume it may consume all the bandwidth that is left available.
4. Links 4 and 5 are robust rerouting links in the presence of link failures. When Link 1 fails, *v1* from Node A will be routed to Node B via Link 4, then delivered to Node D via Link 2. Similarly, Node C will redirect *v2* to Node D via Link 5 and Link 2 in the presence of Link 3 failure.
5. When Link 2 fails, the ABR traffic from Node B will be redirected to Node D through Node A (not Node C).

Note these nodes could communicate with other network components (not shown here) via wired links or other wireless links.

6.2.2 Impact on bandwidth utilization

To examine the impact on bandwidth utilization on a particular SWAN link, we conduct an experiment to measure transmission efficiency when different traffic sources have to be rerouted to share bandwidth of another link in the presence of link faults. In this experiment, the event of fault on Link 1 is at time 15th sec., and later the fault is recovered at time 85th sec. The event of fault on Link 3 starts at time 35th sec., and its recovery happens at time 130th sec. During the link down time, the associated traffic is rerouted to Link 2, based on the decision made by the network routing function.

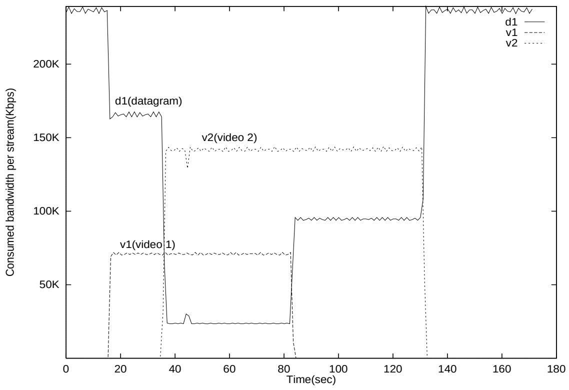

Fig. 8: Received bandwidth in Link 2 using VC-QoS

Fig. 8 shows the effect of link failure to the bandwidth usage on Link 2 with respect to various traffic sessions. At the beginning, traffic *d1* is able to use up all the bandwidth until Link 1 fails. When Link 1 fails, traffic *v1* is rerouted to Link 2 by the routing mechanism. The QoS scheme on Link 2 will then allocate bandwidth used by *d1* to *v1*, since ABR has lower priority than CBR. Similarly, when Link 3 fails, *v2* is granted the required bandwidth after rerouting and *d1* can only use the bandwidth that is left after *v1* and *v2*. In Fig. 8 we also see that *d1* regains bandwidth after the recovery of Link 1 and Link 3.

As a comparison we repeat the same experiment using a non-QoS scheme (UDP/IP). The result is shown in Fig. 9. In this figure, we observe that due to the lack of a QoS mechanism, the amount of bandwidth that a connection can utilize is related to how aggressive the traffic source is. As we have described, *v1* generates data at 73 Kbps, which is much less aggressive than *d1*. Therefore between time 15th and the 85th sec., the quality of *v1* suffers tremendous fluctuations by having to compete with *d1*. Since *v2* is more aggressive (about 145 Kbps) than *v1*, thus between the 35th sec. and the 85th sec., the observed bandwidth shows that all these three sessions get about 1/3 of the bandwidth (Although *v1* is in fact slightly less than the other two). When *v1* stops at time 85th second, *v2* and *d1* both get half of the bandwidth. Also note the fluctuation between time 15th and 35th sec. is more significant than that between time 85th and 130th sec. This is because *v1* transmits video frames slower (0.5 frame/sec) and tends to fall behind the competition with *d1*. However *v2* transmits video at a faster pace (1 frame/sec), so it can share the bandwidth with *d1* more competitively. Moreover, due to the overhead of UDP/IP headers, the maximum bandwidth observed by the receiver here (220 Kbps) is less than previously (240 Kbps) when using QoS VC scheme.

As clearly observed, in this UDP experiment where no QoS is guaranteed, and packets are rerouted correctly after link faults, none of the video sessions get the bandwidth they require. Consequently, they become unattractive in their real-time applications. Furthermore, video session *v1* suffers great fluctuations during its link fault, making the bandwidth it grabs from another link annoying and intolerable to its receivers.

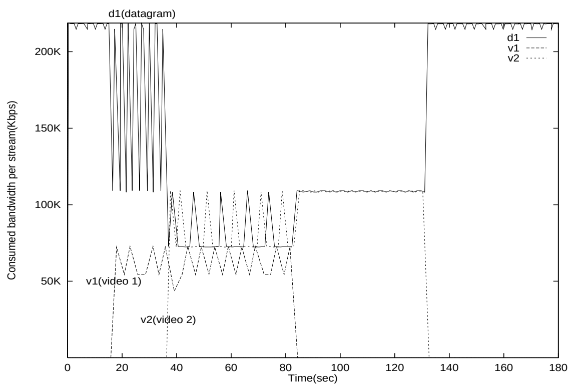

Fig. 9: Received bandwidth in Link 2 using UDP

6.2.3 Impact on Delay Jitter

We also examine the impact to the delay jitter due to a link fault. This time, however, we consider the fault on Link 2 that carries ABR traffic. According to assumption 5 in Section 6.2.1, the routing function in network layer reroutes the ABR traffic from Link 2 to Link 1 after the link fault occurs. The distributions of inter-frame delay of the traffic *v1* carried by Link 1 before and after the traffic rerouting is shown in Fig. 10 and Fig. 11.

Fig. 12 and Fig. 13 present results performed under the UDP protocol for the purpose of comparison. The results are as expected: under the QoS VC scheme, the delay jitter is well controlled even in the presence of the extra traffic due to link fault. On the other hand, using the UDP protocol experiences a completely different result. *V1* has a decent delay distribution before Link 2 fails (Fig. 12); but once Link 2 fails and *d1* are rerouted to Link 1, the delay jitters exhibit uncontrolled and unexpected delays(Fig. 13). Table 2 summarizes the results obtained in Fig. 10 through Fig. 13. We see the UDP (with no QoS) experiences severe delay jitter problems (23.6% overhead) in a heavy traffic situation. Our QoS VC mechanism, on the other hand, is very stable and efficient. Though in Table 2 it seems the QoS VC scheme introduces more overhead (2.6%, computed as the extra delay in relative percentage to the UDP with no background traffic) than the UDP does when no other load is added, overhead introduced by the QoS VC scheme is still less significant then the overhead caused by the TCP/UDP/IP headers.

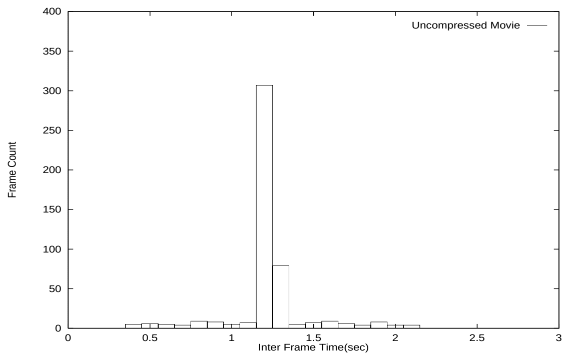

Fig. 10: Before Link 2 fails, with QoS support

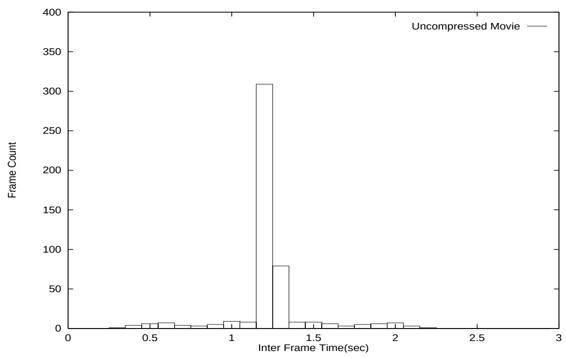

Fig. 11: After Link 2 fails, with QoS support

Transmission scheme	QoS-VC		UDP(no QoS)	
Network load	Heavy	None	Heavy	None
Mean delay (sec)	1.224	1.224	1.474	1.193
Overhead	2.6%	2.6%	23.6%	0%
Variance	0.056	0.056	0.112	0.064

Table 2: Summary of delay jitters observed in two schemes

7 Conclusions

In this paper, we propose a new concept that faults in telecommunications networks often manifest themselves as reductions in service quality, which can be addressed by schemes providing QoS guarantees. We define such a scheme with an API which allows applications to specify the required QoS for a connection. Our QoS VC scheme delivers guaranteed QoS when the radio link is stable.

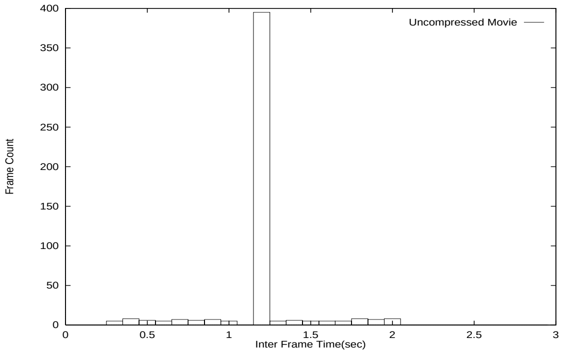

Fig. 12: Before Link 2 fails, without QoS support

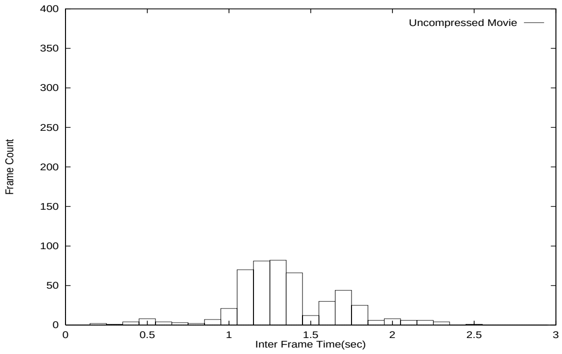

Fig. 13: After Link 2 fails, without QoS support

When the quality changes, the applications gets feedback from this mechanism. Thus instead of an inadequate performance due to insufficient and varying bandwidth, the traffic source has a chance to adjust and best utilize the changing link quality without dropping the connection.

The scheme has been successfully implemented on the SWAN system, with ABR, CBR and VBR supports made available in our current prototype. Modern multimedia applications can be classified into these three categories. The results obtained from the experimental studies on QoS management in the presence of network failures show that our QoS implementation on the wireless ATM network is a valid, efficient, and powerful mechanism which provides guaranteed service quality in an unpredictable, error-prone mobile environment.

Acknowledgements

We would like to acknowledge C. Kintala, P. Agrawal and the SWAN team of Bell Labs. for their support and encouragement of this work.

References

[1] P. Agrawal, E. Hyden, P. Krzyzanowski, P. Mishra, M. B. Srivastava, and J. A. Trotter, “SWAN: A Mobile Multimedia Wireless Network,” in *IEEE Personal Communications*, Apr. 1996. pp. 18-33.

[2] A. Banerjea, Ph.D. Thesis, “Fault Management for Realtime Networks,” University of California, Berkeley, California, December 1994.

[3] D. Ferrari and D.C. Verma, “A Scheme for Real-Time Channel Establishment in Wide-Area Networks,” in *IEEE Journal on Selected Area in Communications*, vol. 8, no. 3, April 1990, pp. 368-379.

[4] D.J. Goodman, “Cellular Packet Communications,” in *IEEE Transactions on Communications*, vol. 38, August 1990, pp. 1272-1280.

[5] M. Holland, G. Gibson, and D. Siewiorek, “Fast, On-Line Recovery Failure Recovery in Redundant Disk Arrays,” in *Proc. 23rd International Symposium on Fault-Tolerant Computing* (FTCS-23), June 1993, pp. 422-431.

[6] M.R. Lyu (ed.), *Software Fault Tolerance*, Wiley, New York, Feburary, 1995.

[7] A. Merchant and P.S. Yu, “Design and Modeling of Clustered RAID,” in *Proc. 22nd International Symposium on Fault-Tolerant Computing* (FTCS-22), June 1992, pp. 140-149.

[8] D. Patterson, G. Gibson, and R. Katz, “A Case for Redundant Arrays of Inexpensive Disks (RAID),” in *Proc. Conference on Management of Data* (SIGMOD’88), June 1988, pp. 109-116.

[9] S. Rangarajan, K. Ratnam, and A.T. Dahbura, “A Fault-Tolerant Protocol for Location Directory Maintenance in Mobile Networks,” in *Proc. 25th International Symposium on Fault-Tolerant Computing* (FTCS-25), June 1995, pp. 164-173.

[10] D.P. Siewiorek and R.S. Swarz, in *Reliable Computer Systems: Design and Evaluation*, Digital Press, 2nd edition, 1992.

[11] C.J. Sreenan and P.P. Mishra, “Equus: A QoS Manager for Distributed Applications,” in *Distributed Platforms*, Publishers Chapman & Hall,1996, pp 496-509.

[12] A. S. Tanenbaum, *Computer Networks* Third Edition, Prentice Hall, 1996.

[13] J. Trotter and M. Cravatts, “A Wireless Adapter Architecture for Mobile Computing,” in *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Comp.*, Apr. 1994, pp. 25-31.

[14] H.M. Vin, P.J. Shenoy, and S. Rao, ”Efficient Failure Recovery in Multi-Disk Multimedia Servers,” in *Proc. 25th International Symposium on Fault-Tolerant Computing* (FTCS-25), June 1995, pp. 12-21.