

# Principles & Experiments of Explicit Delay Control

Xiaoyuan Gu, Dirk Markwardt, Lars Wolf  
Technische Universität Braunschweig  
Mühlenpfordtstr. 23  
Braunschweig 38106, Germany  
{xiaogu,dmark,wolf@ibr.cs.tu-bs.de}

Xiaoming Fu  
Universität Göttingen  
Lotzestr. 16-18,  
Göttingen 37083, Germany  
fu@cs.uni-goettingen.de

## Abstract

*Real-time interactive multimedia applications are highly delay-sensitive, and packets that are out of delay boundaries are usually obsolete. However the current Internet Protocol lacks a way to control the lifetime of the packets explicitly. We propose a packet lifetime control mechanism called Explicit Delay Control (EDC) that embeds a Maximum Tolerable Delay (MTD) field in an IPv4 option. At each network node, the MTD is deducted by the single-hop delay. Packets that expire their lifetime are discarded and non-congestion related delay losses are signaled to the sender to reduce inaccuracy in delay estimations and to adapt to path changes. We implemented EDC in the Linux kernel. Our evaluation has shown that EDC is an effective scheme to ensure the “legality” of the packets, reduce the waste of bandwidth and processing time in the networks, and alleviate congestions.*

## 1. Introduction

The worldwide fever for real-time interactive multimedia applications is soaring thanks to the great proliferation of the Internet and the wide availability of broadband. Most of these applications (e.g. video telephony, networked multiplayer game) are highly delay-sensitive, and depend on the support from the networks. A key requirement is packets that are out of delay boundaries are obsolete to the applications. However the best-effort nature of the Internet poses many challenges for the latency and jitter limits solicited by real-time traffic flows.

The Time To Live (TTL) Field of IPv4 header was designed with two functions in mind [8]: to limit the lifetime of packets, and to terminate Internet routing loops. It is a value expressed in seconds. Each router along the path is supposed to decrement the TTL by one. In reality, TTL is rather a hop count for the maximum number of hops a packet can traverse in the network, than an indication of

the packet’s lifetime. This is the reason why it has been renamed as “HopLimit” in IPv6 [3] to reflect its real intention. Moreover, the time unit used by TTL is second - by far not the resolution level needed for time-critical interactive multimedia applications that normally require a one-way delay within 100ms [9]. Hereby, a new approach from IP itself (apart from QoS mechanisms) has to be in place to play the real role of explicit packet lifetime control, and with high degree of accuracy.

The rest of the paper is structured as follows: We detail the EDC algorithms in Section 2. This is followed by the evaluation in Section 3. The study is summarized in Section 4, with conclusions and the directions for future research.

## 2. Explicit Delay Control Algorithm

To help the readers better understand the terms we will be using in the following sections, we briefly explain the composition of the overall delay that a packet might experience from end to end. It can be coarsely divided into two parts: the end-system delay and the network delay. The former represents the latencies due to the end host’s processing. It usually consists of delays on source coding, compression, packetization, device buffering, channel coding etc. The network delay is composed of propagation delay (only the time for signals to physically propagate on the wires), transmission delay (determined by the network interface with lowest capacity, usually in the last mile), as well as single-hop delay for routing, queuing, and traffic engineering et al. that are part of the PHBs of the routers or middleboxes. [1] gives a thorough analysis of the end-to-end (E2E) delay using video-conference as an example. This is complemented by the work of Papagiannaki et al. [5] on in-depth investigation of single-hop delay at backbone routers.

## 2.1. Algorithm in Brief

We propose a packet lifetime control mechanism called Explicit Delay Control (EDC). The basic idea is to include a one-way Maximum Tolerable Delay (MTD) field in the IP header of each packet belonging to an interactive real-time flow that features a tight delay bound. The MTD is decided by the application, and taking the path round-trip-time (RTT) also into account. At the source host, it is initialized by deducting the estimated propagation delay and transmission delay that the packet will encounter in the network, during session set-up phase. Such estimation is an envision of the relatively constant and predictable portion of the overall delay budget. The MTD is then deducted by the source host processing delay. After the packet enters the network, the MTD is updated by each network node along the path via the per-hop behavior performed by the router. If the lifetime of packet expires at any point, the packet is discarded immediately, as it makes no sense to forward the packet further down the path. Non-congestion related delay losses are signaled to the sender, and help to fine-tune delay estimations and let the sender adapt to path changes.

The benefits of EDC lie in: firstly, it ensures that obsolete packets will be removed from the network at the earliest possible point, and all packets that are consumed finally by the sink are valid in the sense that they comply with the due delays. Secondly, it prevents router(s) and the sink host down the path from spending unnecessary processing time on obsolete packets. Thirdly, the overall network traffic is reduced, since useless traffic is choked down at the earliest possible moment. All of these contribute to the overall system scalability, bandwidth utilization, and help to reduce congestions.

## 2.2. MTD Definition and Initialization

The MTD is a one-way maximum tolerable delay value in millisecond defined by the application as indication of the actual lifetime of the packet. It denotes a delay budget for all possible delay components a packet might experience from the source to the destination. The initialization of MTD at the source host for an established session includes retrieving the application specific delay bound, the measurement of E2E RTT and use half of the result as the approximation of one-way delay. MTD is decided based on these two values. The estimations of propagation delay and transmission delay are followed, complemented by the measure of source host's processing time. The MTD is then initialized by deducting all these three values.

In the calculation of the estimated propagation delay  $EPD$  (see Eq. 1), propagation speed is in practice at two thirds of the speed of the light (200000km/second) for transmitting signals in optical fiber or copper wire. The esti-

mated transmission delay  $ETD$  (see Eq. 2) is related to "the thinnest pipe", which is in most case, the last-mile bottleneck link in the access network. These two results are used to deduct the MTD as described in Eq. 3, together with the measured source processing delay  $SPD$ . This task has to be performed at the end host, as a network node like a router does not have such global view.

$$EPD = \frac{estDistance}{propSpeed} \quad (1)$$

$$ETD = \frac{AvgPktSize \bullet 8}{clientBandwidth} \quad (2)$$

$$MTD = MTD - ETD - EPD - SPD \quad (3)$$

## 2.3. MTD Placement in the IP Header

The IP Option in IPv4 [8] or Extension Header in IPv6 [3] is used to convey the value of MTD. In both cases, MTD is defined as a 4-byte Explicit Delay Control (EDC) IP Option, as described in Fig. 1. The exact number for option type indication will have to be acquired from the Internet Assigned Numbers Authority (IANA) through the standardization procedure. As can be seen, the EDC Option is composed of the option type identification (1 byte), the option length in bytes (1 byte), the TTL (IPv4)/HopLimit (IPv6) value of the IP header at an EDC capable router after the update of the TTL/ HopLimit (explained in Section 2.5), as well as the current MTD value.

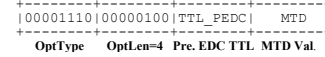


Figure 1. Format of EDC IP Option

## 2.4. Update and Per-hop Behavior on MTD

As can be easily understood, the MTD is rather a dynamic value than a static one, as each network node (e.g. router) along the path is expected to update this value to reflect the remaining lifetime of the packet at that specific time. There are basically two possible ways to renew the MTD. The exact approach is up to the implementation - as a tradeoff between precision and performance.

In the *Absolute Tolerable Delay Update* approach, the router records the time at which the packet arrives at the ingress interface, and before it leaves for the egress interface. The difference of these two is the *Absolute Processing Time* (APT) - the single-hop latency, which contains the table lookup/routing delay, the queuing delay, and any additional overhead if applicable. Should this value be greater

than the current MTD, the packet is discarded. Otherwise, the MTD is updated as  $MTD = MTD - APT$ . The benefit of this approach is the processing time measurement is accurate. The disadvantage is the router will have to spend processing time on the packet even if it realizes in the end that “much ado about nothing” - the packet times out after all the processing.

An alternative is *Proactive Tolerable Delay Update* that anticipates the prospective processing time empirically. On reception of a packet, the router estimates a *Possible Processing Time* (PPT) based on its local knowledge (e.g. the status of its queues). The same decision logic is applied to decide whether to further forward the packet or not. This approach gains processing time on a possibly obsolete packet at the cost of potential inaccuracy in single-hop latency estimation.

In both cases, after a packet has survived such examination, the router updates the prevEDCTTL field in the packet header that tags a latest TTL value at an EDC capable router for handling incompatibilities (described in the next section). After this, the packet is forwarded to the next hop.

## 2.5. Dealing with Incompatibilities

The existence of EDC is detected through parsing the IP Option Field of an IPv4 header or Extension Header of an IPv6 header, for an option type match. However, there might be legacy routers that are not EDC-aware. The solution for this is using the next EDC-capable node to compensate for the missing update(s) of MTD. A missing update of MTD can be detected by checking the TTL field that has been updated at the current router (*currTTL*), against the previous updated TTL value at an EDC-compatible router (*prevEDCTTL*) embedded in the IP option, as  $\Delta TTL = prevEDCTTL - currTTL$ . An example is given in Fig. 2. If the difference of the two satisfies  $\Delta TTL \geq 2$ , a missing update at an EDC incompatible router (router 3 in this case) is detected. Compensation for this lack of update must be performed as:

$$MTD = MTD - CPT \bullet (1 + \beta \bullet (\Delta TTL - 1)) \quad (4)$$

Here, the CPT is current processing time for the packet at the router. The  $\beta$  is the weight, and the value of 1.0 is safely recommended, that means the current router use its own processing time to approximate that of the EDC incapable router to compensate for the missing update of MTD. The prevEDCTTL is updated as  $prevEDCTTL = currTTL$ . In this way, the missing update on the MTD is compensated, following an empirical approach.

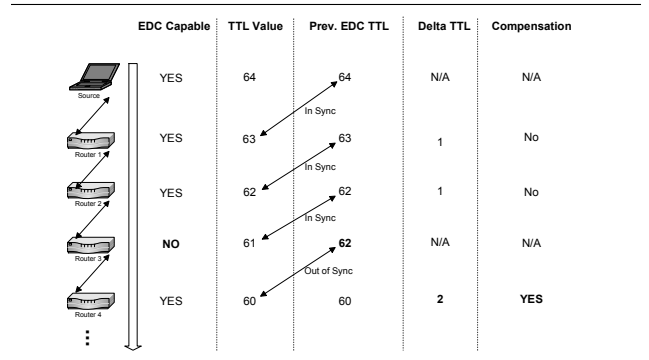


Figure 2. Scenario of TTL out of Sync

## 2.6. Treatment in Tunneling

Tunneling solutions like Mobile IP [7] encapsulates the original IP packet using another IP header with the Care of Address (COA) - the so-called IP in IP Tunneling. The original IP option will be left untouched according to [6], except that the routers will update the TTL of the inner IP Header as if it has undergone normal processing, and the header checksum accordingly. When EDC is used, the inner IP header’s EDC option will also have to be updated as part of the PHBs of the core routers, in addition to the TTL treatment.

## 2.7. Explicit Delay Control Notification

We propose to use ICMP packet to signal to the source, an abnormal loss (non-congestion loss) due to timeout of the delay bound. Congestion related losses are not considered here, since they are usually caused by the overflows of the input buffers of the ingress interfaces of the router. The mechanism is called *Explicit Delay Control Notification* (EDCN). The idea mimics the scenario of TTL reaches 0, in which an ICMP TTL exceeded message is sent to the source. The format of an EDCN message is shown in Fig. 3. The type belongs to the common Time Exceed Type of 11, and with the code number 2 (has to be acquired from IANA). The payload of this ICMP message contains the IP header of the discarded packet, and the first 64 bits of the IP payload that includes the transport layer protocol port number. The EDCN ICMP message is itself carried as payload of an IP packet, sent to the ICMP module at the source host. The upper layer protocol(s) and application get informed correspondingly [2].

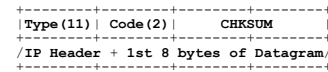


Figure 3. Format of EDCN ICMP message

## 2.8. Sender’s Behavior on Reception of EDCN

Since congestion-related losses are not handled with EDC, a signal of EDCN could have two meanings to the sender: either the initial estimations of the delays as described in Section 2.2 were inaccurate (e.g. as a result of wrong information provided by the user), or the E2E path has changed during the session. While the former happens at the early stage of a session, the occurrence of the later is unpredictable and very rarely (e.g. a link breaks down and the router has to take a redundant link that leads to a different path and hence varied propagation delay). In both cases, adaptation will not be performed should such phenomenon happens simply accidentally, in order to avoid fluctuations in the setting of the MTD. Only if EDCNs repeat for a flow within certain period (typically 4 RTTs), the sender will start the correction.

For the former, a new RTT probing will be carried out to find out the difference (the increase, otherwise a loss will not have occurred) as compared to initial probing. The  $\lambda$  (compensation factor for estimated propagation delay) in Eq. 5 is set to the scale of the increase, while the  $\delta$  (compensation factor for estimated transmission delay) remains unchanged (the bottleneck link is still between the last two hops that decides the estimated transmission delay).

$$MTD = MTD - \left( \frac{ETD}{(1 + \delta)^i} + \frac{EPD}{(1 + \lambda)^i} \right) \quad (5)$$

For the later, the reason can be either an overestimated propagation delay or an overestimated transmission delay, or both. In either case, it has caused the initial setting of MTD being too small (a too tight remaining delay budget), hence a loss in the network. Differentiation can be easily made as the overestimated transmission delay can cause the EDC drop at the last hop, as can be detected by comparing the destination IP address in the packet header to the IP address the current node. Such differentiation is signaled by setting the MTD field of the EDC option of the inner IP header contained in the EDCN message to 255(0xFF) as an indication of wrong estimation of the transmission delay. The exact cause will provide the basis for correction using Eq. 5.

Both  $\lambda$  and  $\delta$  are initialized to 0. The compensation will set the  $\lambda$  to 0.25 for correction of EPD, or set the  $\delta$  to 0.25 for correction of ETD, or both if so desire. This procedure runs for 3 EDCNs belong to the same flow, while the  $i$  denotes the number of tries. As can be expected, after three runs the estimated value nearly doubles as a result of exponential increase.

## 3. Evaluation

In order to verify the kernel and user-space implementations, to proof concept and investigate the performance of EDC, we conducted the following evaluation.

### 3.1. Methodology

The test-bed was realized using User-Mode-Linux (UML) (<http://user-mode-linux.sourceforge.net>). UML enables the emulation of several virtual/logical machines on a single physical Linux host and let them share hardware resources with the real host. UML makes it easy to change the kernel source, recompile it and run it as a virtual machine, so that changes will affect the virtual machine only, without endangering the real host. This speeds up kernel development enormously.

All virtual machines in the test-bed run Debian Linux 3.0 with Kernel 2.4.27. Packet forwarding is enabled on every virtual host. Static routing tables are used so that every host has complete knowledge on how to reach the others. The topology and configuration are shown in Fig. 4. The necessary test programs are stored on the real host and exported via NFS. Network conditions are simulated with the tool Network Emulator (<http://developer.osdl.org/shemminger/netem>). Details of helper programs to calculate physical distances based on city names, and to measure RTT are given in our previous work in (<http://www.ibr.cs.tu-bs.de/arbeiten/xiaogu/aieda.pdf>).

Different from what has been specified in Section 2.7, here, congestion related losses are signaled to the sender for test purpose only. This helps us to collect the information on the number of obsolete packets that are dropped by EDC due to timeouts. Such “unusual” EDCN messages are distinguished from the normal ones by setting the TTL\_PEDC filed of the inner IP header’s EDC option to 255(0xFF).

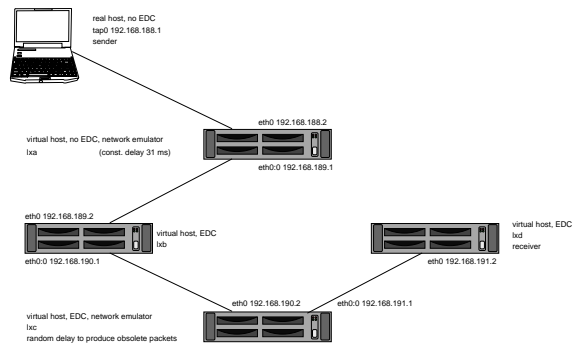


Figure 4. Test-bed configuration

Due to resource constraints that might affect the accuracy of emulated delays at the routers when too many virtual hosts are co-located, we limited the number of hops in our experiments. Hence, some of the routers are cascaded for simplification, as shown in Fig. 4. virtual hosts lxb and lxc are equipped with the network emulator. Host lxa is configured to emulate a constant delay of 31 ms that represents the overall propagation delay. Host lxc is configured to emulate variable delays for different congestion levels. With a probability density function (PDF), it is able to produce 5% of all the packets with queuing delays bigger than their current MTDs, hence cause those packets to be dropped by EDC. Both the network emulator and EDC rely on the timestamp that every IP packet gets assigned at arrival time. The network emulator holds the packet for the duration of the configured single-hop delay. The EDC code will then updates the EDC value with such delay. Host lxa is not EDC enabled so that dealing with incompatibilities can be tested.

### 3.2. Procedure

The server process is started first on host lxd located at “Braunschweig”. The clients are started afterwards with appropriate parameter setting to establish connections. Two test cases have been conducted: Test 1 simulates a transatlantic voice-over-IP connection between Braunschweig and New York with a distance of 6223 km. Test 2 again simulates a voice-over-IP connection, but with a much smaller distance - 1366 km between Barcelona and Braunschweig. All the other parameters remain the same as that used in test 1. In each test case, 9 test runs were carried out, 5000 packets were generated for each test run.

### 3.3. Results

**3.3.1. Test 1: Relaxed timing on simulated transatlantic distance** The client on the real host has been configured with the “New York” as location, 64kbps as link bandwidth, and application profile of “audio”. The client measured an average RTT of 98 ms, and the one way E2E delay was approximated at 49ms . The initial MTD value was set to 100 ms. After subtracting the estimated propagation delay, transmission delay, and source processing delay, the resulting MTD value was 48 ms. The program sent out 5000 data packets and received on average about 230 ICMP EDCN messages. This means that about 4.6% of the packets are detected by EDC due to their MTDs reached 0, compared to the configured obsolete packet proportion of 5%. This gives an accuracy of about 92.0% for EDC to detect the obsolete packets.

**3.3.2. Test 2: Strict timing on shorter simulated distances within Europe** The client on the real host has been

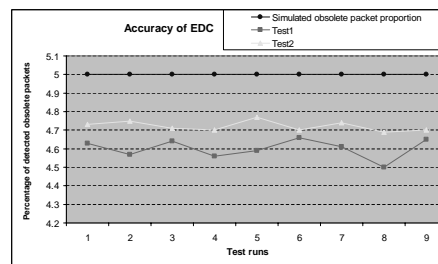


Figure 5. Results of Test 1 and Test 2

configured with “Barcelona” as its location, 64000 bit/s of the link bandwidth and profile of “audio”.

For this test a more strict timing has been defined: The initial MTD value has been set to 40 ms. The delays on the virtual hosts were set to simulate a network path from Barcelona to Braunschweig. This is about 22% of the distance of test 1. Host lxa has a constant delay of 6 ms to reflect the propagation delay. Host lxc is configured to emulate variable delays for different congestion levels, the same way as done in Test 1.

The client found an average RTT of 48 milliseconds, and an approximation of 24 ms was used to represent one-way E2E delay. The initial MTD value was 40 ms. After subtracting the ETD, EPD and SPD, the resulting MTD value was 23 ms.

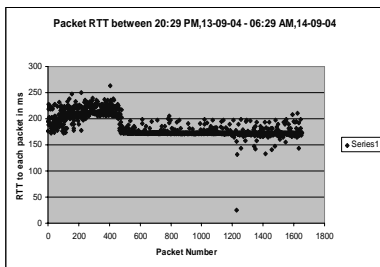
The output of Test 2 is shown in Fig. 5. The program sent out 5000 data packets and received 236 ICMP EDC Time Exceeded notifications. This means that about 4.72% of the packets are detected by EDC due to their MTDs reached 0, compared to the configured obsolete packet proportion of 5%. This gives an accuracy of about 94.4% for EDC to detect the obsolete packets.

### 3.4. Discussions

As validated by our tests, with 5% as the configured proportion of the obsolete packets, EDC is able to effectively detect the obsolete packets at the earliest point in the whole forwarding chain. Unnecessary network traffic and processing power caused by those packets are avoided. Naturally, such gains are visible only if the application delay bounds are tight, and the proportion of the obsolete packets is relatively high. This is true for real-time interactive multimedia applications that cover large physical distances, and congestions in the network occur often. Our previous study on the delay measurement and analysis on cross-atlantic networked games (<http://www.ibr.cs.tu-bs.de/arbeiten/xiaogu/dma4tma.pdf>) has shown that the proportion of out-of-delay-bound packets can be as high as 24.46% in peak hours. Fig. 6 gives an example of a measurement done for a session between a client located

at Braunschweig, Germany and the game server located at the USA, for a one-way delay bound of 100 ms (RTT about 200 ms). In such scenario, EDC can certainly help to save network bandwidth and processing resources.

In comparison, the traditional TTL-based packet lifetime control failed to detect any obsolete packets in both scenarios. The TCP/IP stack implementation of modern operating systems usually initializes value of TTL to 64 or 255. And this value is deducted by one at each network node that the packet passes. The TTL values of packets arrived at their destinations were 49 (for 15 hops) and 45 (for 19 hops) respectively for test1 and test2. Obviously, due to the fact that seldom will a packet traverse more than 30 hops in the Internet, and TTL does not care about the time spent on packet processing, transmission and propagation, the packet lifetime control function of TTL is actually handicapped, which leaves its usage only as hop-count for terminating routing loops. In summary, the TTL limits the distance/hops an IP packet can go whilst the EDC controls the real lifetime of a packet.



**Figure 6. Measurements of round-trip-time for game traffic**

One possible concern about the proposed EDC mechanism is the overhead it might bring to the backbone routers. According to study in [5], packets with IP options spend about  $36 \mu\text{s}$  inside a router, which indicates that the absolute processing time on IP option is in the magnitude of 36 microseconds at a single router, and multiplied by the number of traversed routers along the path. This is rather minor overhead for today's mainstream backbone routers. Another concern is whether the approach is destructive rather than constructive. This to a large extent depends on the delay distributions of the traffic and the characteristic of the application. If the proportion of obsolete packets is too high, the application should be able to terminate the service for certain users based on the EDCN. Furthermore, it hinges on to what extent the service can still be usable with the losses of the outdated packets. Yet another issue is some routers and firewalls tend to give packets with IP options some 'spe-

cial' treatment (e.g. passing them through the slow path or putting them into some penalty box). However this is not universal. Protocols such as RSVP(-TE) and IGMPv2 have followed also router alert option approach, but have been widely accepted and implemented by the operators with high processing efficiency at the backbone routers. We certainly hope and strive to develop EDC into this direction.

## 4. Conclusions and Future Work

We have proposed in this paper a new packet lifetime control mechanism for IP-based real-time multimedia applications. To the best of our knowledge, there is no such mechanism present in literature. The proposed Explicit Delay Control scheme is valuable for most real-time interactive applications that desire the consumption of only packets within certain delay boundaries, and in environment where the proportion of obsolete packets are high. The benefits sourced from EDC can be easy understood, as multimedia traffic is purified with the obsolete packets being filtered out at the earliest possible points, and the downstream network bandwidth and processing time on those stale packets are avoided. In addition, the proposal also contributes to system scalability and congestion control.

Beside what we have done in this work, we have identified a number of future extensions. First, further improvement of the precision of MTD (e.g. using  $\mu\text{s}$  instead of ms) to better reflect the single-hop latencies for the processing at fast routers. Second, a better approach to determine the user location is needed. Information provided by the users has potential trust and accuracy problems. Simply relying on city names denotes another limitation (e.g. duplicated city names representing cities are geographically completely differently located). For example, Geobytes (<http://www.geobytes.com>) offers the service of finding a physical region for a given IP address. Third, experiment with the real world routers for computational overhead and further performance evaluation of EDC are on our shortlist, so are porting and optimization. Forth, the current approach of determining the one-way E2E delay is a simplification. It certainly has to be improved by taking the asymmetric paths in the Internet due to BGP policy routing into account, which can cause the forwarding path to be decoupled from the reverse path, hence different delays. Last but not the least, joint-work with QoS mechanisms. EDC is not on an isolated island, a reasonable combination of QoS schemes (e.g. Diff-Serv, Int-Serv, or Int-Serv over Diff-Serv), or certain priority/class-based queuing mechanisms [4] with EDC will certainly further enhance QoS. For example, at the ingress point of the router, should the Proactive Tolerable Delay Update as described earlier detects a possible expiration on MTD, instead of discarding the packet,

assigning it to a high priority queue will possibly rescue the packet.

With the presented work in this paper and with possible extensions, we hope EDC will play the real role of “Time To Live”, and contribute to the prevalence of killer IP-based multimedia applications.

## Acknowledgment

This work has been partly supported by the European Union under the E-Next Project FP6-506869.

## References

- [1] M. Baldi and Y. Ofek. End-to-end delay analysis of videoconferencing over packet-switched networks. *IEEE/ACM Transactions On Networking*, 8(4):479–492, Aug 2000.
- [2] D. Comer and B. Stanford. *Internetworking with TCP/IP Vol.1 Principles, Protocols, and Architecture 4th Edition*. Prentice Hall, Upper Saddle River, NJ, 2000.
- [3] S. Deering and R. Hinden. Internet Protocol, Version 6 (Ipv6) Specification. RFC2460, Dec. 1998.
- [4] B. B. et al. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC2309, Apr. 1998.
- [5] K. Papagiannaki, S. Moon, C. Fraleigh, and C. Thiran, P. and Diot. Measurement and analysis of single-hop delay on an ip backbone network. *IEEE JSAC*, 21(6):908–921, Aug 2003.
- [6] C. Perkins. IP Encapsulation within IP. RFC2003, Oct. 1996.
- [7] C. Perkins. IP Mobility Support. RFC2002, Oct. 1996.
- [8] J. Postel. The Internet Protocol. RFC791, Sept. 1981.
- [9] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of Networking in Multiplayer Computer Games. In *Proceedings of ADCOG21*, pages 74–81, Hong Kong, China, Nov 2001.