

Promote the Use of Explicit Delay Control

Xiaoyuan Gu, Dirk Markwardt, Lars Wolf

Institute of Operating Systems & Computer Networks, Technische Universität Braunschweig

Mühlenpfordtstr. 23, Braunschweig 38106, Germany

Email: {xiaogu,dmark,wolf}@ibr.cs.tu-bs.de

Abstract—The Internet is undergoing changes of its traffic mix, with the IP-based interactive multimedia applications gaining momentum. According to studies, UDP-based multimedia traffic has in recent years risen to above 25% of the overall Internet traffic volume, as compared to only 5% a decade ago. To these delay sensitive real-time interactive multimedia applications, out of delay boundary packets are usually obsolete. However the current Internet Protocol lacks a way to control the lifetime of the packets effectively. In this paper, we propose the Explicit Delay Control (EDC) as a viable packet lifetime control mechanism designed for such purpose. It uses an embedded Maximum Tolerable Delay (MTD) field in an IPv4 option. The value of MTD is determined by taking the application specific delay bound, measured round-trip-time, and the estimations of the path propagation delay and transmission delay into account. 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 can ensure the “legality” of the packets, reduce the waste of bandwidth and processing time in the networks, and help to alleviate congestions.

Index Terms—packet lifetime control, multimedia, network entertainment, quality of service

I. 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 [1]: 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 [2] 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 [3]. 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.

In this paper, we propose the Explicit Delay Control (EDC), as a viable packet lifetime control mechanism designed for such purpose. It uses an embedded Maximum Tolerable Delay (MTD) field in an IPv4 option. The value of MTD is determined by taking the application specific delay bound, measured round-trip-time, and the estimations of the path propagation delay and transmission delay into account. 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 can ensure the “legality” of the packets, reduce the waste of bandwidth and processing time in the networks, and help to alleviate congestions.

This is an extended work on our previous conference paper [4] in four aspects: i) A dedicated related work section that covers the recent works on delay analysis and monitoring and the works on quality of service has been added. ii) A new section on the implementation has been included. iii) More test results are presented. iv) The conclusion and future work section has been better detailed.

The remaining part of the paper is structured as follows: We start with a survey of the related work in Section 2. Then the EDC algorithms is introduced in Section 3. Section 4 presents the internals of the Linux kernel implementation of EDC. This is followed by the evaluation in Section 5. The study is summarized in Section 6, with concluding remarks and directions for future research.

II. RELATED WORK

A. Delay: basics, composition, analysis and monitoring

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. The total delay budget can be coarsely divided into two parts: the end system delay and the network delay.

The former represents latencies due to the processing at the end systems. It usually consists

of delays on source coding/decoding (e.g. for compression/decompression of audio/video contents, and likely involves also buffering), data encapsulation/decapsulation, packetization/depacketization, header compression/decompression (if any), channel coding/decoding (if any), device buffering as well as any possible overhead of the protocol stack. CPU scheduling can also cause additional delay on any time-sharing system.

The network delay is composed of such components like propagation delay (the absolute time for the signals to physically propagate on the wires at a speed of usually $2/3$ the speed of light), transmission delay (determined by the hardware bandwidth of slowest link or in other words, the smallest capacity of all network interfaces, which is usually in the last mile), single-hop delays for routing (table lookup), queuing, and traffic engineering that are part of the PHBs at the routers, processing at the middle-box (e.g. firewall, NAT box, proxy, cache), as well as any other possible delays in the network.

The works on delay monitoring can be traced back to the IP Timestamp Option [5], which monitors catenet user traffic for one way delay, with the prerequisite of the synchronization of the network devices. The first visible work on delay monitoring started with Bolot [6], by using UDP probing packets with variable intervals to measure RTT and losses in the Internet. Hsiao et al. in their work in [7] proposed the active delay control for long-lived TCP traffic that by collecting congestion signals at the routers, to adjust the traffic load balancing among concurrent flows to reduce the chance of timeouts. [8] gives a very thorough analysis of the end-to-end delay using video-conference as an example. This is complemented by the work of Papagiannaki et al. [9] on in-depth investigation of single-hop delay at backbone routers. Choi et al. [10] proposed a one-way delay monitoring method within an ISP - a fragment of the end-to-end path. However, it requires for direct access to the core routers, and the synchronization of the clocks of the routers, which can be a prohibiting task.

B. Quality of service

By its origin, the Internet is a packet-switched network designed for delivering packets in a best effort fashion, e.g., routers perform routing and forwarding of IP packets without distinguishing them from each other. With the booming of the IP-based multimedia applications such as voice over IP or video conference, there is a strong need for providing Quality of Service (QoS) for end-to-end applications.

QoS usually refers to the mechanisms for meeting application-perceived network performance requirements, such as bandwidth, delay, jitter and loss rate. It requires a number of fundamental changes to the Internet architecture, including the introduction of new components for (control plane) resource reservation and (forwarding plane) traffic control etc., in order to provide service differentiation [11].

Due to ever-increasing capacity of communication networks and devices, over-provisioning has been proposed in the recent years. Some backbone operators or regions with surplus bandwidths consider it as an alternative to QoS provisioning. However, high bandwidth does not necessarily imply a guaranteed QoS such as delay assurance [12], even if bandwidth can increase more quickly than the soar of network traffic. Therefore, the topic of QoS for IP networks has represented a critical agenda for over a decade.

The IETF IntServ model [13], in connection with the Resource Reservation Protocol (RSVP) [14], [15], introduced a per-flow QoS provisioning model. End-to-end QoS is provisioned by allocating dedicated bandwidth to the premier subscribers via end-to-end signaling using RSVP. In this model, an end-to-end traffic flow, which is usually characterized as a token bucket, once have passed admission control, needs to install and maintain per-flow reservation states in IntServ nodes. Such mechanism is usually combined with a classifier and proper queuing and scheduling schemes, as defined in [16]–[18]. However, IntServ is widely criticized for its scalability due to the fact that the routers have to maintain per-flow states. No 100% guarantee of the desired delay can be achieved, as bandwidth is a sufficient condition to reach the delay bound, not a necessary condition. Also, such scheme limits its clientele, as for the system, bandwidth will soon be used up with the increases of Int-Serv users.

To address the scalability problem of per-flow QoS provisioning, the DiffServ [19] architecture and the related per-hop forwarding path behaviors [20], [21] were proposed. DiffServ provides a looser control of end-to-end flows, mainly depends on the configured PHB associated with the service class as specified in the DiffServ Code Point (DSCP) field of the IP header. However, even with DiffServ, it could be possible that network is severely congested an even a packet with the highest priority simply can not pass through. In addition, how to provide QoS across boundaries between different DiffServ domains remains an open issue. In this context some work have proposed, e.g. IntServ over DiffServ networks [22] and the bandwidth broker model [23].

On the other hand, packet queuing, scheduling and priority dropping are other proposals to achieve QoS. There have been queuing disciplines other than best effort (FIFO) version, for example, Round Robin (RR), Weighted Fair Queuing [WFQ] [24], priority-, class-based or hierarchical queuing [25]–[27], and various active queuing or dropping techniques [28]–[31]. They allow a more flexible resource management than FIFO in providing different levels of QoS guarantees. Notably, when active queuing is used, hosts do not have to rely on buffer overflow as the only indication of congestion. Instead, a router can use explicit congestion notification [32] to indicate congestion to hosts.

Unfortunately, a common weakness of QoS provisioning is not all users are willing to pay for it or can afford the additional costs of QoS, especially those non-

enterprise subscribers who care pretty much about the figures on their bills. ISPs and operators have also hesitated for long at QoS, due to the concerns on complexity, compatibility and scalability. It would be good to have a simple scheme to let the application to customize its own delay constraint without additional monetary costs to the end users, for better utilization of the network resources. This is exactly what we are proposing in this paper.

III. EXPLICIT DELAY CONTROL ALGORITHM

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.

A. 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 t_{PROP} (see Eq. 1), propagation speed is the absolute time for signals to physically propagate on the wires over the distance d_{DIS} which in the case of optic fiber medium, 2/3 of the speed of the light ($c_{LIG} = 3 \cdot 10^8$ m/s), and ϵ is the weighting factor of the type of the transmission medium.

$$t_{PROP} = \frac{d_{DIS}}{\epsilon \cdot c_{LIG}} \quad (1)$$

is related to "the thinnest pipe", which is in most case determined by the network interface with lowest capacity c_{NIC} - usually the last mile when the end-users are hooked up to the Internet, given the size of the packet s_{PACK} . This is expressed in Eq. 2.

$$t_{TRAN} = \frac{s_{PACK} \cdot 8}{c_{NIC}} \quad (2)$$

The t_{PROP} and t_{TRAN} are then 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.

$$t_{MTD} = t_{MTD} - t_{PROP} - t_{TRAN} - t_{SPD} \quad (3)$$

B. MTD Placement in the IP Header

The IP Option in IPv4 [1] or Extension Header in IPv6 [2] 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 III-D), as well as the current MTD value.

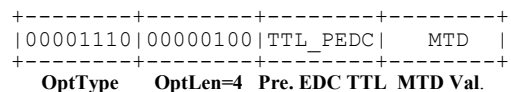


Fig. 1. Format of EDC IP Option.

C. Update and Per-hop Behavior on MTD

As can be easily understood, the t_{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* t_{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 t_{MTD} is updated as $t_{MTD} = t_{MTD} - t_{APT}$. 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* t_{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 v_{PTTL} 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.

D. 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 v_{CTTL} that has been updated at the current router, against the previous updated TTL value v_{PTTL} at an EDC-compatible router. An example is given in Fig. 2. If the difference $\Delta = v_{PTTL} - v_{CTTL}$ is greater than 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:

$$t_{MTD} = t_{MTD} - t_{CPT} \cdot (1 + \beta \cdot (\Delta TTT - 1)) \quad (4)$$

Here, the t_{CPT} is current processing time for the packet at the router. The β 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 v_{PTTL} is updated as $v_{PTTL} = v_{CTTL}$. In this way, the missing update on the MTD is compensated, following an empirical approach.

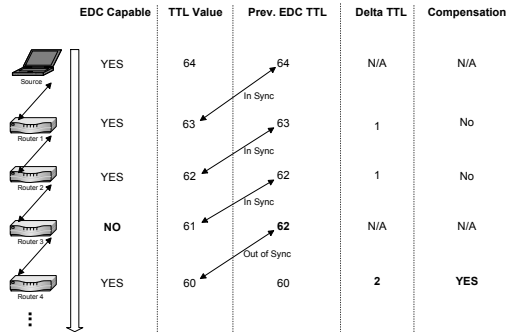


Fig. 2. Scenario of TTL Out of Sync.

E. Treatment in Tunneling

Tunneling solutions like Mobile IP [33] 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 [34], 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.

F. 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 [35].

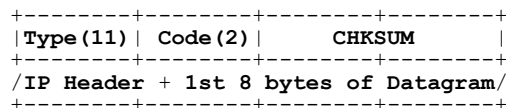


Fig. 3. Format of EDCN ICMP Message.

G. 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 III-A 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 rare (e.g. a link breaks 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 λ (compensation factor for estimated propagation delay) in Eq. 5 is set to the scale of the increase, while the δ (compensation factor for estimated transmission delay) remains unchanged (the bottleneck link is still between the last two hops that decides the estimated transmission delay).

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

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 λ and δ are initialized to 0. The compensation will set the λ to 0.25 for correction of EPD, or set the δ 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.

IV. IMPLEMENTATION

In this section, we describe the implementation of EDC algorithm in the Linux kernel as well as the adoption of some helper programs in greater details.

A. EDC Handling in the Linux Kernel

The implementation of EDC was realized using the Linux operating system. It is chosen because its open source code allows us to customize the kernel according to our needs. With the “User-Mode-Linux” (UML) tool [36], “virtual hosts” are instantiated. Hence, it is possible to carry out kernel development within such user space

without endangering the main host. After compiling the UML-Kernel of the virtual hosts, a virtual host can simply be started within user-space of the main host. As can be imagined, even coding bugs will not crash the whole system. UML is further explained in Section V-A.

1) *Modifications to the Kernel Source:* In order to realize EDC in the Linux kernel, it is necessary to make the corresponding adaptations at multiple source files as listed below:

- icmp.h: An additional code block for ICMP packet type “Time Exceeded” has been defined for EDC with the value of 2.
- icmp.c: When generate ICMP packets, the Linux kernel supports a ICMP packet size up to 576 bytes, including the IP header and payload. According to the ICMP Section of RFC 1122 [37], only the IP header and the first eight bytes of the payload should be used to form an ICMP packet. As EDC is intended to save bandwidth, the size of ICMP EDC Time Exceeded packets is limited to a minimum possible size. 4 bytes is used in the current implementation of EDC, as part of the IP header option.
- inetdevice.h, sysctl.h and devinet.c: The structure of parameters for a network interface has been extended by two values: real_speed and mtd_decrement. The latter is for testing purposes and can be dropped later.
- ip.h: A new code for IP EDC Option has been added to the existing IP option types. Until an exact number for this purpose has been assigned by the Internet Assigned Numbers Authority (IANA), the code number 14 will be used. The internal IP options structure has been modified to accommodate the MTD value and the EDC TTL Field.
- ip_forward.c: The test code for the parameter “mtd_decrement” is located in the source file ip_forward.c. An IP packet with EDC option will be dropped if its MTD value is smaller than or equal to the mtd_decrement value of the network interface the packet would leave. Otherwise the decremented MTD value will be written back into the packet, a new IP checksum will be calculated and the packet will be enqueued for sending. These code changes are needed for testing and can be removed later.
- ip_options.c: Here, the additional fields of the internal IP options structure needed for EDC get filled.
- ip_output.c: Every IP packet the Linux kernel receives gets a timestamp. To measure how long a locally created packet stayed in the send queue, it is necessary to implement this timestamp also at packet creation time.
- sch_generic.c: If an IP packet with EDC option is dequeued from the network interface send queue, the Absolute Processing Time t_{APT} will be calculated. Additionally, the amount of time which will be needed to send out this packet on the given network interface regarding packet size and real upstream bandwidth (with the parameter “real_speed”) will also be calculated.

If the difference between the IP TTL and the EDC TTL values is greater than 1 the packet passed one or more EDC incompatible hops. In this case the EDC TTL Field will be set to the value of the IP TTL Field. The processing time calculated above will be multiplied with the number of EDC incompatible hops. The missing update of the MTD Field caused by the EDC incompatible router(s) will be compensated by assuming the same processing delay measured above.

If the MTD value of the packet is less than or equal to the sum of these two time values, the packet will be dropped and an ICMP EDC Time Exceeded packet will be sent back to the source of the original packet. Otherwise the MTD Field within the packet will be updated considering the number of EDC incompatible hops, and the IP checksum will be recalculated.

2) *Configuration of the EDC-enabled Kernel:* This can be done via the sysconfig interface. There are two parameters which affect EDC for every network interface. They can be found within the proc-filesystem: `/proc/sys/net/ipv4/conf/*/ {mtd_decrement,real_speed}`

- `mtd_decrement` The “mtd_decrement” parameter is needed to emulate the network interface queuing delay. If an IP packet with EDC option leaves the system through a network interface with a valid MTD, the value of MTD will be additionally updated by deducting the queuing delay of the current networking interface.

Example:

```
echo "10" }
/proc/sys/net/ipv4/conf/eth0/mtd_decrement
```

 will decrement the MTD value by an additional amount of 10 milliseconds if the IP packet leaves the system via eth0.

- `real_speed` This is the parameter to configure the link bandwidth of a network interface. The unit is bits per second (bit/s). The setting of this value will affect the estimated transmission delay should the current interface is the thinnest pipe in the whole path.

The below example illustrate the configuration of the eth0 interface with a bandwidth of 128kbps :

```
echo "128000" } /proc/sys/net/ipv4/conf/eth0/real_speed
```

B. User-Space Application

The user-space test application consists of a server and a client. Both are written in C.

1) *Server:* The implementation of the server is rather simple. The server is the data sink and its main task is binding to a UDP socket and listen for incoming traffic. The administrator of the server knows where the hardware is located. Clients can connect this server from all over the world. It is reasonable to leave the task of estimating the propagation delay to the server because only one central location database should be maintained.

If the server receives an UDP packet containing the string “getproppdelay:xxx” where “xxx” is the name of the city the client is located, it calls the helper program

“distance.pl” passing its own location and the location it received from the client as parameters. The estimated propagation delay will be sent back to the client using a single byte.

Starting the Server:

The server process must be started with its location as an argument, the below example starts the server process with its location “New York”.

`./server "New York"`

2) *Client:* The client program is named as “udptest”, and it has substantially greater amount of tasks than the server. Its main purpose is to send test data to the server using IP packets with EDC option enabled.

The following tasks must be performed first: Determining the round-trip-time (RTT) between the client and the server by querying the helper program “rttometer”, estimating propagation delay by querying the server, estimating transmission delay using the given bandwidth, and opening a raw socket for receiving ICMP EDC Time Exceeded notification packets.

Test data is generated using three profiles: “audio”, “video” and “game”. Profile “video” assumes that a normal video stream with 25 frames per second, each frame has a size of 200 bytes. The other two profiles have been measured by examining the data streams of normal applications (Table IV-B.2).

For the profile “game”, the data stream of the First Person Shooter “Unreal Tournament 2004” (UT2004) has been examined. During normal session, the client sends 20 packets per second with an average size of 75 bytes to the server.

For profile “audio” the data stream of the Internet Telephony application “Skype” has been examined. A call to the Echo Test Service resulted in 23 Packets per second with an average size of 136 bytes.

TABLE I
PACKETS PER SECOND AND AVERAGE PACKET SIZES IN BYTES OF
DIFFERENT MULTIMEDIA APPLICATIONS

	Video	Audio (Skype)	Game (UT2004)
Packets/s	25	23	20
Avg. size	200	136	75

During sending test data according to one of the three profiles, the client application takes care of analyzing ICMP packets that the client host receives. This is necessary because an ICMP raw socket receives all ICMP packets addressed to the particular host even if they are not related to the running application. Furthermore the estimated transmission and propagation delay values will be adapted using the appropriate correction factors if the circumstances apply to the description in the EDC algorithm.

Starting the Client:

The client program expects 4 parameters: destination host, own location, own upstream bandwidth and profile to be used. The parameter “own location” must be the name of the city where the client is located, it must be enclosed with quotation marks if the name contains

spaces. The “own upstream bandwidth” must be given in bit/s, “profile” can be one of “audio”, “video” or “game”. The below example starts the client with the destination host 192.168.191.2, the own location “Braunschweig”, an upstream bandwidth of 33600 bit/s and the profile “game”.

C. Helper Programs

In order to keep the server and client simple, it is necessary to use several helper programs.

1) *distance.pl*: The small Perl script “distance.pl” calculates distance and propagation delay between two locations on earth. It is called from the server process with the two locations as arguments. It fetches the latitude and longitude values of these locations from the Cities-database from Geobytes [38]. The distance is calculated using the equation 6 given in [39].

$$\begin{aligned}
 distance = \arccos(& \cos(lat1) * \cos(lon1) * \\
 & \cos(lat2) * \cos(lon2) + \\
 & \cos(lat1) * \sin(lon1) * \\
 & \cos(lat2) * \sin(lon2) + \\
 & \sin(lat1) * \sin(lat2)) * \\
 & radius
 \end{aligned} \quad (6)$$

The “radius” variable has the value of 6378 km which is the radius of the earth. Locations to the west of the zero meridian and to the south of the equator have got negative latitude / longitude values. Equation 6 calculates the shortest possible distance between the two locations.

For the calculation of propagation delay, it is assumed that the transmission media is fiber optics. Light speed in vacuum or copper wire is ca. 300000 km/s (C), but in the medium glass it is only about 200000 km/s. In case of an error, the estimated propagation delay is set to 0. This can happen when one or both locations can not be found in the database. So it is not possible to subtract a too big estimated propagation delay from the initial MTD which could cause unintended packet drops due to errors in the estimation.

One problem of “distance.pl” is that it can not distinguish between multiple cities which have the same name, e.g. the city name “Hamburg” appears 10 times in the database. Internally, the script simply takes the first one it finds. This limitation is acceptable since these test programs are only a proof of concept. Possible improvement on this is pointed out in the future work section.

2) *rttometer*: The Program “rttometer” [40] measures round trip time using TCP. This avoids problems occurred when firewalls blocking ICMP Echo Requests traffic should the standard “Traceroute” program is used.

A raw socket is used to send TCP SYN packets to the destination host. With the arrival of either a TCP ACK or TCP RST packet, the RTT is calculated. The measurement is repeated several times to get an statistically accurate value.

The client program calls “rttometer” to measure the RTT from client to server. The RTT value is needed for deciding whether an adaption of the estimated transmission and propagation delay values must take place.

The one-way end-to-end delay is approximated as half of the measured RTT. This value is then used as reference to correct a misestimate of one-way propagation delay or transmission delay.

V. 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.

A. 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. The approach is different from VMWare [41] and Bochs [42]. The latter emulate complete X86 PCs with its own graphic hardware, memory and interfaces, while UML lets the virtual machines and let them share hardware resources with the real host. The kernel of the UML virtual host will be compiled for the special hardware platform “UM” and runs as a normal user-space program without the need for superuser privileges. Image files on the real host will be used as virtual hard disks. Networking is done via the TAP device and a “Switch” Daemon which emulates an Ethernet switching device. 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 III-F, 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 field of the inner IP header’s EDC option to 255(0xFF).

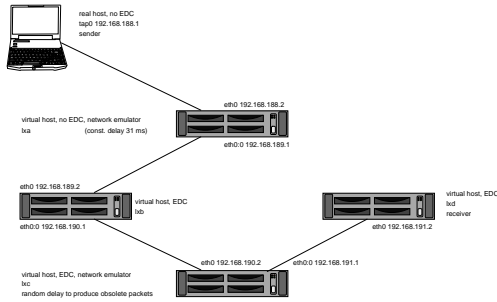


Fig. 4. Test-bed Configuration.

B. Network Emulator

The network emulator used in this work is part of the kernel distribution. The development of Network Emulator [43] was inspired by the “NISTNet” [44]. Network emulation will be done by the special “netem” scheduler. The emulation parameters will be assigned by the “tc” command from the iproute2 package. It is possible to emulate constant and variable delay, random packet loss, duplication and packet reordering. This tool has been included in the mainstream Linux kernels since version 2.4.27. 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.

C. 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.

D. Results

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.

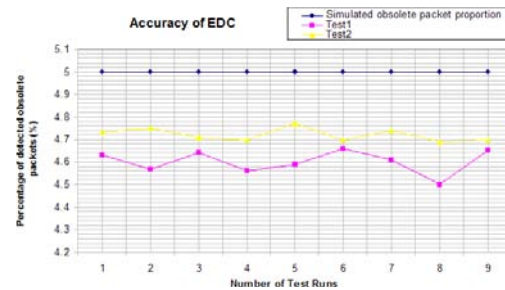


Fig. 5. Results of Test 1 and Test 2.

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

E. 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 about 25% in peak hours. Fig. 6 and Fig. 7 give two example of measurements done for session between a client located at Braunschweig, Germany and the game servers located at the USA and Canada, 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.

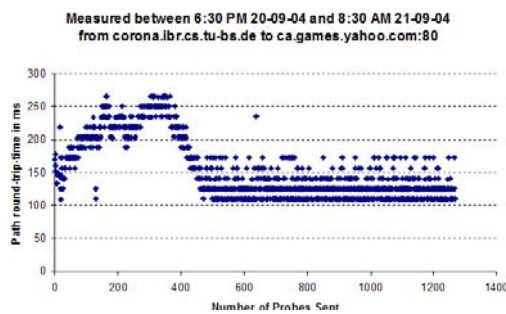


Fig. 6. Measurements of Round-trip-time for Game Traffic 1.

One possible concern about the proposed EDC mechanism is the overhead it might bring to the backbone routers. According to study in [9], packets with IP options spend about 36 μ 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



Fig. 7. Measurements of Round-trip-time for Game Traffic 2.

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 'special' 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.

VI. CONCLUSIONS AND FUTURE WORK

We have proposed in this paper a new packet lifetime control mechanism for IP-based multimedia applications with high time resolution. 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 applications that desire the consumption of only packets within certain delay boundaries, and in environment where the proportion of obsolete packets are high (e.g. equal or more than 5%). 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.

We presented the implementation details of EDC in Linux kernel and described the evaluation methods and procedures. In both test cases, EDC has demonstrated acceptable accuracy in dealing with obsolete packets. The result will be even more significant in cases where obsolete packets have a larger proportion of the Internet traffic mix. Internet service providers and big IP carriers might be interested in EDC, especially nowadays many end Internet users subscribe to a flat rate service. Any form of bandwidth saving technique without too much complexity and overhead will be appreciated.

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 μ s instead of ms) to better reflect the

single-hop latencies for the processing at fast routers. In the current design, the time unit for MTD is milliseconds that a single byte is used for this purpose. However, IP options must be 32 bit aligned and must end with the “Option-End” opcode 0. Therefore in the current EDC implementation, the total space needed for the EDC option is 5 bytes. To stick to 4-byte word rule, and to avoid the 3-byte padding (see Figure 8), a larger space for better precision can be used. Also, Explicit Congestion Notification bits for non-TCP and non-DCCP flows can also be co-located here.

Option Code Content: 14 (EDC)	Option Length Content: 4 (4 Bytes)	EDC TTL Content: variable	EDC MTD Content: variable
Option End Content: 0 (End)	Padding Content: 0 (Fill Byte)	Padding Content: 0 (Fill Byte)	Padding Content: 0 (Fill Byte)

Fig. 8. Actual Structure of IP Option With Only EDC Used.

For example, a possible improvement could be to increase the resolution of the MTD by expand this field from one to four bytes, which will make it possible to store 2^{32} values instead of 2^8 values. This would be nanoseconds, compare to milliseconds, without additional length for such enhancement (see Figure 9).

Option Code Content: 14 (EDC)	Option Length Content: 4 (4 Bytes)	EDC TTL Content: variable	EDC MTD Content: variable
EDC MTD Content: variable	EDC MTD Content: variable	EDC MTD Content: variable	Option End Content: 0 (End)

Fig. 9. Better Accuracy of MTD and No Padding Bytes.

- Second, a better approach to determine the user location is needed. The method mentioned earlier to determine the user locations and calculate the distance between the end points for propagation delay is fine for test purpose. In reality, there are uncertainties in the information provided by the end users. Trust and accuracy are also problems. Simply relying on city names denotes the other limitation (e.g. duplicated city names representing cities are geographically completely differently located). The Geobytes [38] tool and SmartWhois [45] tool, possibly others too, offer the service with which it is possible to find the region where an IP address is located. This can be used to estimate distance, hence the propagation delay without the inputs from the users. Incorporating a similar algorithm into the overall design is a must for EDC. Attention should also be paid to the end system located behind a Network Address Translation (NAT) box, although the distance between the two is usually marginal. A

solution for this could be using the real IP address of the NAT box or gateway instead.

- Third, experiment with the real world routers: A third work item is the experiment with the real world routers. Backbone routers are not running Linux, e.g. routers from solution provider like Cisco are working with proprietary Cisco IOS. Computational overhead and performance evaluation on those products are necessary to examine the gain from EDC. Porting is a logical step for such work. Optimization is also critical.
- Forth, better method for the determination of one-way delay. 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.
- Fifth, joint work with QoS. 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 [28] 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] J. Postel, “The Internet Protocol,” RFC791, IETF, Sept. 1981.
- [2] S. Deering and R. Hinden, “Internet Protocol, Version 6 (Ipv6) Specification,” RFC2460, IETF, Dec. 1998.
- [3] J. Smed, T. Kaukoranta, and H. Hakonen, “Aspects of Networking in Multiplayer Computer Games,” in *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, Hong Kong SAR, China, November 2001, pp. 74–81.
- [4] X. Gu, D. Markwardt, X. Fu, and L. Wolf, “Principles and Experiments of Explicit Delay Control,” in *Proceedings of IEEE Consumer Communications and Networking Conference(CCNC'06)*, Las Vegas, USA, January 2006.
- [5] Z.-S. Su, “A Specification Of The Internet Protocol (Ip) Timestamp Option,” RFC781, IETF, May 1981.
- [6] J.-C. Bolot, “End-to-end packet delay and loss behavior in the internet,” in *Proceedings of the ACM SIGCOMM 1993*, San Francisco, USA, September 1993, pp. 289–298.
- [7] P.-H. Hsiao, H. T. Kung, and K.-S. Tan, “Active Delay Control for TCP,” in *Proceedings of the IEEE Globecom 2001*, San Antonio, USA, November 2001.
- [8] M. Baldi and Y. Ofek, “End-to-end delay analysis of videoconferencing over packet-switched networks,” *IEEE/ACM Transactions On Networking*, vol. 8, no. 4, pp. 479–492, Aug 2000.
- [9] 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*, vol. 21, no. 6, pp. 908–921, Aug 2003.

- [10] B.-Y. Choi, S. Moon, R. Cruz, Z.-L. Zhang, and C. Diot, "Practical delay monitoring for ISPs," in *Proceedings of the 2005 ACM conference on Emerging network experiment and technology (CoNext)*, Toulouse, France, October 2005, pp. 83–92.
- [11] X. Xiao and L. Ni, "Internet qos: The big picture," *IEEE Network*, vol. 13, no. 2, pp. 8–18, 1999.
- [12] K. Nahrstedt, "To overprovision or to share via qos-aware resource management?" in *Proc. of IEEE International Symposium on High Performance Distributed Computing*, Aug. 1998.
- [13] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an overview," RFC1633, IETF, June 1994. [Online]. Available: <http://www.ietf.org/rfc/rfc1633.txt>
- [14] L. Zhang, S. Deering, D. Estrin, S. Shen, and D. Zappala, "Rsvp: A new resource reservation protocol," *IEEE Network*, vol. 7, no. 5, pp. 8–18, Sept. 1993.
- [15] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Integrated Services in the Internet Architecture: an overview," RFC2205, IETF, Sept. 1997. [Online]. Available: <http://www.ietf.org/rfc/rfc2205.txt>
- [16] J. Wroclawski, "The use of RSVP with IETF integrated services," Internet Engineering Task Force, RFC 2210, Sept. 1997. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2210.txt>
- [17] —, "Specification of the controlled-load network element service," Internet Engineering Task Force, RFC 2211, Sept. 1997. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2211.txt>
- [18] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," Internet Engineering Task Force, RFC 2212, Sept. 1997. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2212.txt>
- [19] S. Blake, D. Black, D. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC2475, IETF, Dec. 1998. [Online]. Available: <http://www.ietf.org/rfc/rfc2475.txt>
- [20] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding PHB group," Internet Engineering Task Force, RFC 2597, June 1999. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2597.txt>
- [21] B. S. Davie, A. Charny, J. W. R., K. B. Benson, J. Y. Le, W. Courtney, S. Davari, and V. Firoiu, "An expedited forwarding PHB (per-hop behavior)," Internet Engineering Task Force, RFC 3246, Mar. 2002. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc3246.txt>
- [22] Y. Berner, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, and B. S. Davie, "A framework for integrated services operation over diffserv networks," Internet Engineering Task Force, RFC 2998, Nov. 2000. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2998.txt>
- [23] K. Nichols, V. Jacobson, and L. Zhang, "A two-bit differentiated services architecture for the Internet," Internet Engineering Task Force, RFC 2638, July 1999. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2638.txt>
- [24] J. Bennett and H. Zhang, "Worst-case fair weighted fair queueing," in *Proc. of INFOCOM*, 1996.
- [25] H.-Y. Wei and Y.-D. Lin, "A survey and measurement-based comparison of bandwidth management techniques," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 2, pp. 10–21, 2003.
- [26] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks," in *Proc. of SIGCOMM*, 1998.
- [27] I. Stoica, H. Zhang, and T. Ng, "A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 185–199, 2000.
- [28] B. Braden, D. D. Clark, J. Crowcroft, B. S. Davie, S. E. Deering, D. Estrin, S. Floyd, and V. Jacobson, "Recommendations on queue management and congestion avoidance in the Internet," Internet Engineering Task Force, RFC 2309, Apr. 1998. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2309.txt>
- [29] W. Feng, D. Kandlur, D. Saha, and K. Shin, "The blue queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 513–528, 2002.
- [30] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.
- [31] S. Floyd and K. Fall, "Promoting the use of the end-to-end congestion control in the internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, 1999.
- [32] K. G. Ramakrishnan, S. Floyd, and D. L. Black, "The addition of explicit congestion notification (ECN) to IP," Internet Engineering Task Force, RFC 3168, Sept. 2001. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc3168.txt>
- [33] "IP mobility support," Internet Engineering Task Force, RFC 2002, Oct. 1996. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc2002.txt>
- [34] C. Perkins, "IP Encapsulation within IP," RFC2003, IETF, Oct. 1996.
- [35] D. Comer and B. Stanford, *Internetworking with TCP/IP Vol.1 Principles, Protocols, and Architecture 4th Edition*. Upper Saddle River, New Jersey: Prentice Hall, Mar. 2000, ISBN:0130183806.
- [36] J. Dike, "User-mode Linux Kernel (UML)," <http://user-mode-linux.sourceforge.net>.
- [37] "Requirements for Internet hosts - communication layers," Internet Engineering Task Force, RFC 1122, Oct. 1989. [Online]. Available: <http://www.rfc-editor.org/rfc/rfc1122.txt>
- [38] "Geobytes Inc.: Geoworldmap database containing cities of the world with geographical coordinates," <http://www.geobytes.com>.
- [39] C. Michaels, "Formula and code for calculating distance based on two lat/lon locations," http://jan.ucc.nau.edu/~cvm/latlon_formula.html.
- [40] A. Zeitoun, "Rttometer," <http://idmaps.eecs.umich.edu/rttometer>.
- [41] "VMWare Inc.: Vmware," <http://www.vmware.com>.
- [42] K. Lawton, "Bochs," <http://bochs.sourceforge.net>.
- [43] S. Hemminger, "Network emulator," <http://developer.osdl.org/shemminger/netem>.
- [44] "National Institute of Standards and Technology, The NIST Net Network Emulator," <http://www-x.antd.nist.gov/nistnet>.
- [45] "TamoSoft Inc.: SmartWhois," <http://www.tamos.com/products/smartwhois>.

Xiaoyuan Gu is a Ph.D. student at the Institute of Operating Systems and Computer Networks in the Technical University of Braunschweig, Germany. He received his M.Sc. with honors from the International University in Germany in Information and Communication Technology in 1999. During September 2001 and September 2003, he was with the Panasonic Multimedia Communication European Lab as a research engineer. His research interests include autonomic communications, network architecture, cross-layer design, and networked entertainments. Xiaoyuan is a member of the Autonomic Communication Forum (ACF), and student member of IEEE Communications Society (IEEE ComSoc) and the Association of Computing Machinery (ACM). He chairs the IEEE Interest Group on Autonomic Communications (ACIG) and has served on the TPC of many international conferences such as IEEE ISM'05, IEEE CCNC'06, IEEE ICTTA'06, IEEE ICC'06 etc. Xiaoyuan is a recipient of the Year 2005 "Chinese Government Award for Outstanding Oversea Ph.D. Student".

Dirk Markwardt is a master student at the Computer Science Department of the Technical University of Braunschweig. His main research interests include real-time applications and QoS in mobile and wireless communications.

Lars Wolf is professor of Computer Science. He is head of the Institute of Operating Systems & Computer Networks and the dean of the Faculty of Mathematics and Computer Science at the Technical University of Braunschweig. Dr. Wolf received his diploma (M.Sc.) degree in 1991 and the doctoral degree in 1995 both in computer science. He was a research staff member at the IBM European Networking Center, Heidelberg, Germany between 1991 and 1996. He led the Multimedia Networking Research Group in the Department of Electrical Engineering and Information Technology at Darmstadt University of Technology between 1996 and 1999. Dr. Wolf became an associated professor at the Institute of Telematics in University of Karlsruhe in December 1999, till he joined Technical University of Braunschweig in April 2002. Professor Wolf has authored

and co-authored 8 books and book chapters, and published more than 90 papers. Dr. Wolf serves on chair, TPC member of numerous conferences and workshops. He is editorial board member of Elsevier Journal of Computer Communications, Baltzer Journal of Telecommunication Systems, Kluwer Journal of Multimedia Tools and Applications, and the Journal of Praxis in der Informationsverarbeitung und Kommunikation (PIK). Dr. Wolf is a member of IEEE, ACM and ACF.