# On improving data transmission in networks

*Sur l'optimisation de la transmission de données dans les réseaux*

Habilitation thesis in computer science
*Habilitation à diriger des recherches dans la spécialité informatique*
Université de Franche-Comté

**Eugen Dedu**

**Committee:**

| | | |
|---|---|---|
| *Rapporteurs:* | Christian Müller-Schloer | Leibniz Universität Hannover, Germany |
| | Toufik Ahmed | LaBRI / ENSEIRB-MATMECA, Bordeaux |
| | Hervé Guyennet | FEMTO-ST / Université de Franche-Comté, Besançon |
| *Examiners:* | Emmanuel Lochin | ISAE, Toulouse |
| | Julien Bourgeois | FEMTO-ST / Université de Franche-Comté, Montbéliard |

Defended on 3 December 2014
*Soutenue publiquement le 3 décembre 2014*

Montbéliard, France

# Contents

# Introduction

This document presents the research activities I have been conducting since september 2003, when I was hired as assistant professor (*maître de conférences*) and started to work in a new field, computer networks (previously I had worked on parallelism and multi-agent systems in other places in France). These activities have taken place in the same laboratory, namely Institut FEMTO-ST, DISC computer science department, previously named LIFC laboratory, and in the same place, in Montbéliard, France.

During this period, I have co-supervised several Ph.D. students: three graduated, one not graduated, and two ongoing students. I also supervised three master's thesis.

The document presents only the work I have been directly involved of, i.e. either by myself or by Ph.D. students I co-supervised.

This work has been published in 7 international journals and 16 international conferences with peer reviewing. Most of them are regarded as "referenced" publications (i.e. high quality) in our laboratory. The articles are available on my Web page[1].

### Manuscript organisation and personal contributions

My research work is in the domain of network communication. If I look back at all my published work concerned by this manuscript, I would divide it in four fields. In this document, each field is presented in one chapter. Chapters are presented in chronological order of my interest in them, and sections inside are presented in chronological order too. Even if sometimes sections could also be presented in other order, the chronological order is perfectly appropriate.

Ideas presented in the first chapter, congestion control in networks, are general and very varied, and each section presents a completely different idea; as such, the first chapter is the biggest in number of pages. It presents some ideas to increase data transmission speed on wireless networks and on networks in general, and also an analysis of congestion control in sensor networks.

The second chapter, adaptive video streaming, focuses on video transmission, and presents three main independent ideas: a video adaptation method, a method to reduce oscillations in quality, and an analysis of video adaptation methods found in the literature.

The third chapter presents the work done inside a project funded by the national agency of research (ANR) on MEMS communications. In this chapter, each section depends on previous ones, and each section can be considered as a next step in the work. They present a framework to discover the best differentiation criteria, a method to find out the best size of the platform, implementation on the real platform, an enhanced differentiation method and a tree-based storage of the knowledge.

The fourth chapter is the shortest chapter and presents very recent and ongoing research on nanonetwork communications. It presents one published article, on nanonetwork simulation, and another one under review, presenting a method to reduce the energy when transmitting data.

Except perhaps the first part of the third chapter (which is particular to a platform), the idea which crosses all the work is optimisation or improvement, hence the title of the manuscript. I have not discovered new materials, I have not created new devices; what I have tried is to optimise their use. For example, one can still be astonished that in our ultra fast network era seeing a simple Web page still takes a few seconds; my proposition in this case was to prioritise flows so that short flows such as HTTP have higher priority in the network [58]. Another example is that devices in nanonetworks use energy to transmit 1 bits, and no energy to transmit 0 bits; my proposition was to encode data transmitted so that it contain fewer bits 1 [203].

[1] `http://eugen.dedu.free.fr`

**Thanks**

I would like to thank Julien Bourgeois, our current team leader. He helped me by introducing me in the Smart surface project (the object of chapter on distributed MEMS), introducing me to various conference committees, working with me in various fields, and, sharing the same office with me, by answering to my uncountable questions about the work and research in general.

I thank the jury members for accepting to be in the jury and for spending their time with this manuscript.

I thank people with whom I co-published articles, especially the Ph.D. students I co-supervised, and all people whose discussion allowed me to improve my research knowledge.

# Chapter 1

# Congestion control in networks

## 1.1 Introduction

When I joined the current laboratory as assistant professor in 2003, the research team worked on computer networks and on video transmission. The first field I worked on was naturally computer networks. My colleagues worked at network and transport levels. After some discussion with my team leader (François Spies), I decided to focus on transport level.

In the TCP/IP networking model, the transport level is on top of network layer, and below application layer. The network layer takes care at transmitting one packet from source to destination. Each packet has two IP addresses: source and destination. Network layer uses destination IP address to route the packet from machine to machine up to the correct destination.

In a network, making a packet reach destination is not sufficient. IP is a best-effort protocol, which means that it does its best to make packet reach to destination, but does not provide any guarantee on its arrival. Additionally, packets can be duplicated and when sending several packets, they can arrive in disorder. More importantly, when a source needs to send a big quantity of data, such as an e-mail of 1 MB or a Web page of 100 kB, it must cut the data in packets of about 1500 bytes and send each packet independently. The limit of 1500 bytes is given by the technology of the link between machines (the so-called MTU, Maximum Transmission Unit). The role of congestion control at the sender is to provide the timing of sending the packets. Sender should not send them too slowly, for example 1 packet/second, because the network would be underutilised. Sender must not send them too fast either, for example all the packets as fast as possible, because network could very likely become congestioned, make packets be lost and prevent other communications to take place.

Research work on transport level in general networks has been focusing mainly on congestion control. The main protocols used at transport level were TCP and UDP. TCP has congestion control, whereas UDP does not. As such, TCP is much more used than UDP. During that time (around year 2005), a new protocol, DCCP, which has congestion control too, was being standardised by IETF (the group managing the protocols used in Internet). My research direction turned on congestion control in general, and simulations/experiments on TCP and the then-new protocol DCCP.

Congestion control is performed by data sender, but involves the network also. The machines inside the network which take care of routing packets are the routers. In brief, routers receive packets on their interfaces, look at their destination, and route them nearer destination to a neighbouring router through another interface. If a router receives more packets than it can handle or can store in its memory, then it drops some packet, so the packet becomes lost. Several mechanisms exist on deciding when and which packet the router should drop. Once the packet is lost, the packet sender usually discovers the lost packet and retransmits it.

My work in this field resulted in various ideas on congestion control:

- Various ideas on generic congestion control on computer and sensor networks: optimise packet loss in routers [122], prioritise short flows (e.g. Web flows) on routers [58], analyse congestion control influence in sensor networks [54].

- Optimisation on wireless networks: remove MAC retransmission influence on wireless net-

works [57] (which will not be presented in this document), differentiate between wired and wireless packet losses, initially proposed in [158] and improved and with more experiments in [163].

## 1.2 Optimisation of packet loss in routers

The network quality of service (QoS) and the congestion control of the transport protocol are important parameters for the performance of a network data transfer. To this end, routers use various queue policies for packet dispatching, and all of them must deal with packet drop. This section proposes a new algorithm for packet drop in routers. Given that a packet drop *wastes* all the network resources it has already used, we propose a new policy which favours packets with higher distance from source. This policy, DDRED (for Distance-Dependent RED), consists in dropping packets having consumed fewer physical resources and thus coming from sources nearer to the congested router. By dropping the packets from this source, retransmission becomes faster. Simulations show that long flows are indeed favoured compared to short flows, and lead to higher overall resource utilisation without sacrificing flow fairness.

### 1.2.1 Introduction

During data transmission, routers may become congested. In such cases, some packets must be dropped from the input queue(s). Several policies exist in order to decide which packets will be rejected. Two very common such policies are DropTail and RED [78].

In DropTail, a new packet is rejected if and only if the queue is full. It is a very fast decision and hence it is suited to backbone routers.

RED (Random Early Detection) [78] is an active queue management currently implemented in many routers. Using RED leads to better sharing among the various flows passing through the router. RED is also used for congestion management through negative feedback to the sender, which is done by dropping packets before queue overflows in order to signal imminent congestion. If utilization of ECN is enabled in the router and flow is ECN capable, RED marks these packets instead of dropping them. To do it, RED maintains a few values: queue length $q_l$, queue average $q_{ave}$, minimum queue threshold $q_{th\_min}$ and maximum queue threshold $q_{th\_max}$.

- If $q_{ave} < q_{th\_min}$, all packets pass without being dropped or marked.
- If $q_{ave}$ is between $q_{th\_min}$ and $q_{th\_max}$, packets are marked (if ECN) or dropped (if not) with a probability which increases linearly while $q_{ave}$ increases.
- Finally, when $q_{ave} > q_{th\_max}$ all packets are dropped.

Packet management is performed by queue policies in routers, such as the two above. Our proposition is that when the policy specifies that the incoming packet should be dropped, drop another packet instead of the incoming one. DDRED is the implementation of our proposition in RED policy, but it can be integrated into other policies, such as DropTail.

### 1.2.2 Related work

There are several techniques to achieve higher overall resource utilisation and/or to adapt the application requirements to the dynamic network conditions. Some techniques act on the host side, such as the congestion control of TCP [152], others on the router side. In the following we focus on techniques on the router side.

Furthermore, some techniques are content-based [37]. For example, in an MPEG video streaming flow, routers recognize the type of frames (I, P or B) and try to prevent I frame dropping (which are critical, since P and B depend on I).

Several techniques which are not content-based exist. Some of them use various scheduling algorithms, such as FQ (Fair Queue) and WRR [175] (Weighted Round Robin).

Others use various queue management policies, such as RED and its derivatives. Cisco's WRED [46] (Weighted RED) includes IP preference in RED by providing separate thresholds and weights for different IP addresses. DSRED [205] (Double Slope RED) improves the performance of RED by dynamically changing the slope of the RED probability curve as a function of the congestion level.

In [92] several queue sizes with different parameters are used when the queue is scarcely filled, leading to a faster RED processing. MRED (Modified RED) [3] optimises RED for bursty traffic. Both use NS2 to validate their proposition.

In best-effort networks, equipments do their best to deliver packets, but there is no guarantee that they will arrive at destination, neither can one be sure of the time they will take to arrive. DiffServ [22] is a method which tries to guarantee a QoS on such networks, by dividing traffic into groups with different priorities on routers.

AECN [207] (Adaptive ECN) adds to the TCP header a field containing information about the RTT (Round Trip Time) of a flow. The field is set by senders and read by routers. Routers have a set of RTT ranges and corresponding flow sub-queues. Each packet is put in the appropriate sub-queue, based on its RTT. Unfortunately, the RTT gives the return time and is *independent* of the location of the packet in its trip, while in DDRED the distance gives the number of routers (resources) involved *up to* the router which would drop the packet.

### 1.2.3 Adaptive priority mechanism

In a point of the network, we define the distance $d_r$ of a packet as being the number of routers between this point and the source of the packet. If a packet is to be dropped, it is preferable to drop a packet with a small $d_r$ value and to keep the packet with a large $d_r$ value. We use this distance to develop new types of queue management policies.

#### Principle

Bearing this in mind, we can use the TTL field of 8 bits of the IP header [151]. The TTL is the lifespan of a packet in a network, a kind of expiry date. Each time the packet enters a router, its TTL is decremented, and when it reaches the critical value of zero, the packet is destroyed, even if it has not yet reached its destination. This field is initialised at wish by senders [151], and its initial value cannot be known by routers. Therefore, we need our own IP fields, implemented as IP options for example. We present two methods to implement these fields.

The first method is to take into account the distance $d_r$. This can be done using one additional field in the IP header: the initial TTL ($TTL_i$). $TTL_i$ is set in each packet with the initial value of the classical TTL field of the *source* machine. By computing the difference between this new field and the TTL read on the router, we can determine that the distance $d_r = TTL_i - TTL$ covered by the packet.

The second method is to take into account the percentage of the route covered. It involves two fields: $TTL_i$ and $TTL_f$, the latter (final TTL) being set with the TTL read on the destination of the previous packet.

As the second method requires more resources (CPU time and one more byte in the IP header), we choose here the first method in our simulations.

#### Implementation

The router uses the covered distance to choose the packet to be dropped. We implement our mechanism in two common queue management policies used in routers:

In DropTail policy the $d_r$ value of an arriving packet and of the last $N$ packets of the queue can be compared. The one whose distance is smaller, therefore pertaining to the flow having the nearer source, is rejected, whereas the other is put at the tail of the queue.

In RED policy the decision of packet drop (leading the router to a congestion state) is based on a probability computing as presented in previous section. We do not change the formula but only the packet to which this probability applies. When the probability requires dropping (or marking in

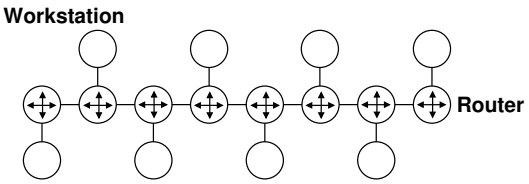



Figure 1.1: Bus network topology.

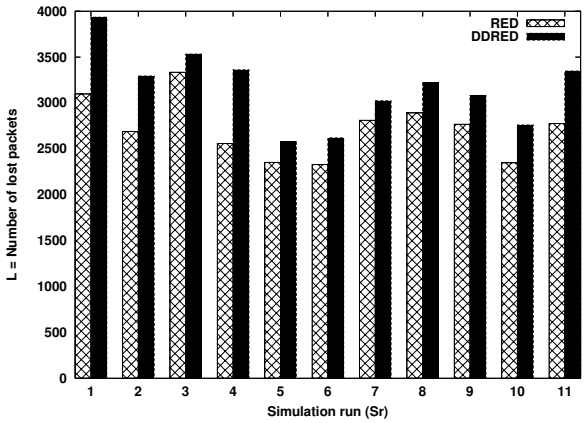



Figure 1.2: Bus topology, number of lost packets.

ECN case) the packet, the router searches the queue for the nearest packet (smallest $d_r$) and drops it instead of the incoming packet.

### 1.2.4 Simulation results

For our simulations, we use Network Simulator [140] version 2.29. We create a patch which changes the drop policy in the RED algorithm as presented before. For more reliability, each case study (named $Sr_n$ for Simulation run $n$) has been simulated with different initial random seeds (from 0 to 10), giving different scenarios. Each scenario is simulated twice with the same initial conditions (transfer size and transfer starting time): in the first one, all routers implement the RED algorithm and in the second one the DDRED algorithm. All the routers use either the RED policy or the DDRED one and we compare the results based on packet losses, which give information about network resource utilisation.

We first simulate a simple network, afterwards a complex network, and finally we discuss the fairness of the mechanism proposed.

**Simple network** The network is shown in figure 1.1. All the links between workstations and routers and between routers have a bandwidth of 10 Mbits/s. 100 FTP transfers, based on TCP New Reno, are created in a period of 60 seconds and the simulation stops when all the transfers are completed. The source and the destination of each of these flows are randomly selected among the workstations. The size of the data transferred is randomly selected (but reproducible) between 100 kB and 6 MB.

Figure 1.2 presents the number of lost packets during the entire simulation. The average number of lost packets is $L_{RED} = 2724$ and $L_{DDRED} = 3160$. It shows that there are more losses with DDRED. Having more losses is not a bad thing. What is important is not that a packet is lost but that it has consumed resources (router processors and bandwidth), for example a packet lost at the 10th router compared to 3 packets lost at the second router.

To measure the resources consumed, for each distance covered $d_r$ the number of packets lost $L_d$ is totalled and then the number of losses is weighted by their respective covered distance:

$$S = \sum_{d_r=1}^{d_{rmax}} (d_r \times L_d) \tag{1.1}$$

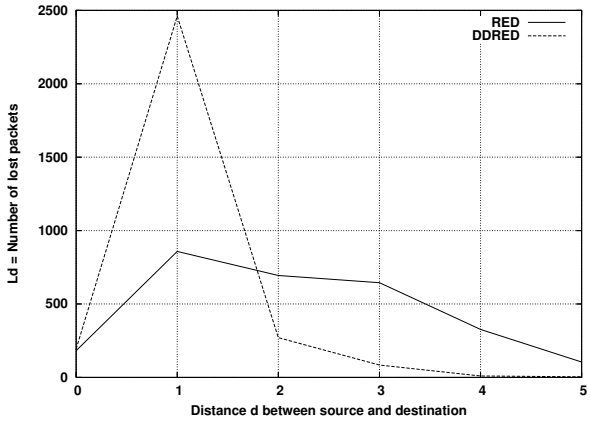



Figure 1.3: Bus topology, loss distribution for a simulation run ($Sr_6$).

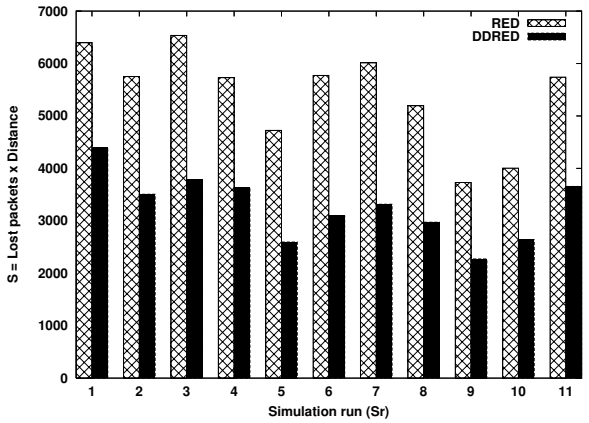



Figure 1.4: Bus topology, number of losses weighted by their covered distance.

Figure 1.3 shows an example of loss distribution $L_d(d)$ for one of the scenarios. As seen, DDRED lost packets are drawn nearer to the source of the transfer. Figure 1.4 shows the sum for each simulation run $Sr_n$. We thus save space in the router queue or "slots" (a slot is the space of one packet with average size in the queue). The average consumed slot number is reduced from $S_{RED} = 5418$ to $S_{DDRED} = 3259$.

Simulations also show that the sum of transmission times of all the flows is 2.5% smaller with DDRED than RED, which means that flows arrive faster to destination.

**Complex network** In order to have a more realistic scenario, a more complex topology, with a shape of *flower*, is used in the next simulations. According to a small study of the xDSL backbone of a Internet provider[1], most of these networks are built around a central core where several loops are connected. These loops are composed of a small number of routers. The aim of the closed loop is to have a fault tolerance.

We considered a flower with five loops with eight routers on each of them. As in the bus network, only one workstation is connected to a router and connections have the same conditions and parameters (sizes and times of transfers). In this simulation 500 connections are initialized in the same first minute and the simulation ends at the end of the last transmission, as in the first one.

Here only the results which differ from the previous simulation are presented. Figure 1.6 presents the number of packets lost. Contrary to the bus network, fewer packets are lost, approximately 22% ($L_{RED} = 48083$ and $L_{DDRED} = 37402$).

[1] `http://support.free.fr/reseau/province.png`, not available anymore as of August 2008; however `https://www.renater.fr/IMG/png/Carte_2014.png` is similar, with several centres instead of only one.

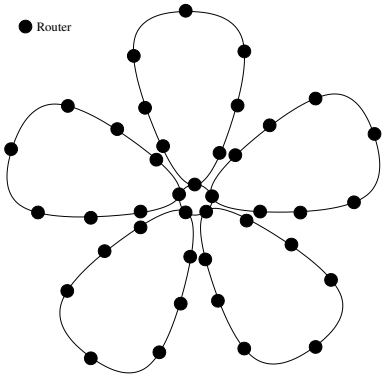



Figure 1.5: Backbone of the *flower* network.

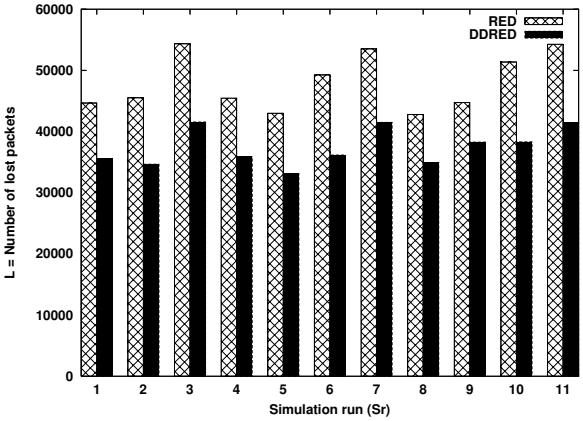



Figure 1.6: *Flower* topology, number of lost packets.

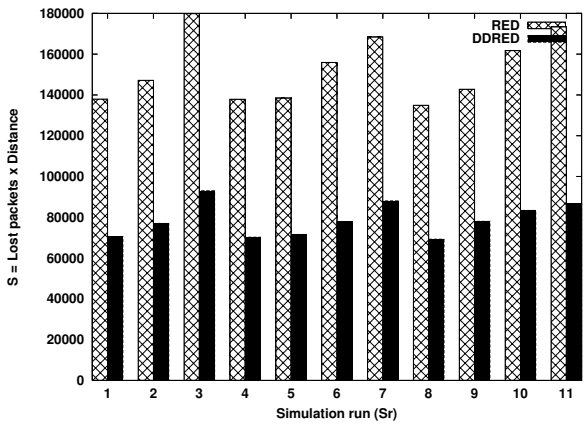



Figure 1.7: "Flower" topology, number of losses weighted by their covered distance (slot saving).

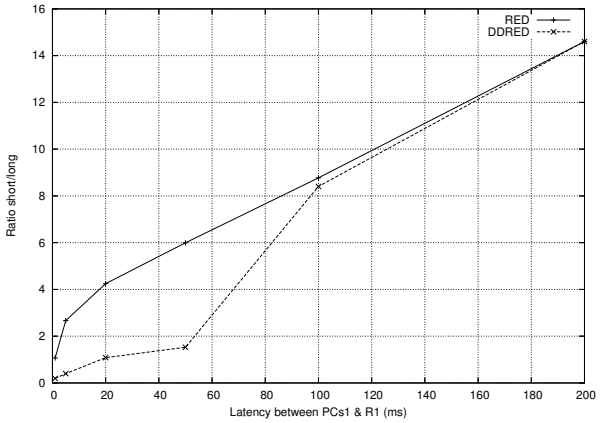



Figure 1.8: Ratio between short and long flow bandwidth.

In figure 1.7, the number of used "slots" with these queue policies is on average $S_{RED} = 152595$ and $S_{DDRED} = 78683$ packets. With DDRED, the saved "slots", therefore available for other flows, can grow up to 50% compared to RED.

The sum of the transfer times is on average 6% smaller. Not all the transfers are faster, but on average transfers end sooner.

**Fairness discussion** To compare the fairness of DDRED to RED, the ratio of bandwidth between the two flows is analysed and simulated. Figure 1.8 shows the results of one simulation. In this figure, the nearer to 1 the ratio is, the fairer the algorithm is. (As already known, the ratio between the bandwidths of two TCP flows is not exactly 1, but depends on their RTT, see for example [134].) The figure shows that for a difference of latency inferior to 10ms (i.e. difference of RTT inferior to 20ms), RED has a ratio nearer to 1, hence RED is fairer. However, between 10ms and 100ms, DDRED has a ratio nearer to 1, hence DDRED is fairer. When the latency is greater than 50ms, the fairness of both strategies is the similar. As a conclusion, DDRED works better than RED for networks where latencies vary much, which is the case for Internet.

### 1.2.5 Conclusions

During congestion, routers drop packets, no matter the queue policy. When a packet is dropped, all the network resources it has consumed are *wasted*. This section proposed a new algorithm of packet drop on router, which takes into account the path covered by a packet up to this router. Packets far from their sources are favoured compared to packets near their sources. Simulations show that packets from long path flows are favoured, and network resources are less used without sacrificing the TCP fairness.

## 1.3 Prioritisation of short flows on routers

This section presents a study on the benefits of favouring the transfer of packets of a TCP flow over a best-effort network such as IP. Specifically, we aim at studying whether we could improve the pace of short data request, such as HTTP request, by giving a high priority to TCP packets that are not previously enqueued inside a core router. Following the idea that long-lived TCP flows greatly increase the routing queue delay, the motivation of this work is to minimise the impact in terms of delay, introduced by long-lived TCP flows over short TCP flows. Thus, this forwarding scheme avoids to delay packets that do not belong to a flow already enqueued inside a router in order to avoid delay penalty to short flow. We define metrics to study the behaviour of such forwarding scheme and run several experiments over a complex and realistic topology. The results obtained present interesting and unexpected property of this forwarding scheme where not only short TCP flows take benefit of such routing mechanism.

### 1.3.1 Introduction

Favouring the TCP connection establishment packets or any others packets belonging to a TCP flow inside a core network is not a novel idea. James Kurose, in his famous text book Computer Networking, suggests that it would be useful to protect from losses TCP packets with a low time-to-live value in order to prevent retransmission of packets that have already done a long travel inside a core network. As another example and in the context of QoS networks, Marco Mellia et Al. [137] have proposed to protect from loss some key identified packets of a TCP connection in order to increase the TCP throughput of a flow over an AF DiffServ class. In this study, we observe that TCP performance suffers significantly in the presence of bursty, non-adaptive cross-traffic or when it operates in the small window regime, i.e. when the congestion window is small. The main argument is that bursty losses, or losses during the small window regime, may cause retransmission timeouts (RTOs) which will result in TCP entering the slowstart phase. As a possible solution, we propose qualitative enhancements to protect against loss: the first several packets of the flow in order to allow TCP to safely exit the initial small window regime; several packets after an RTO occurs to make sure that the retransmitted packet is delivered with high probability and that TCP sender exits the small window regime; several packets after receiving three duplicate acknowledgement packets in order to protect the retransmission. In this study, we are focused on the TCP throughput guarantee which is a primary goal of the DiffServ/AF class [22, 88]. This motivates to protect against losses, packets that strongly impact on the average TCP throughput. In an obvious manner, we cannot guarantee a minimum throughput over a best-effort network. However, we propose to study the prioritisation of certain TCP packets in order to investigate whether we could minimise the transfer delay of short TCP flows without impacting on the long lived connections. In this paper, we study how to exploit router functionality to improve the performance of TCP flows in a best-effort network by giving a higher priority to the first packets of a TCP flow inside a router queue if no others packets belonging to the same flow are already en-queued. One of the main goals of this proposal, called FavourTail, is to investigate and understand the benefits of using a prioritisation forwarding scheme inside a core network. Intuitively, we expect to decrease the transfer delay of small TCP connections but surprisingly, we show that this routing behaviour does not only improve the performance of TCP flows and allows to improve the overall performance in terms of transfer delay when slight congestion occurs. We present the potential benefit of using such solution and analyse the benefit of our scheme over a realistic and complex network topology. We show that the proposed scheme can improve the transfer delay of TCP flows up to 25%.

### 1.3.2 Related work

Many papers present router-based methods to optimise flow transfers. This section presents the most related to our FavourTail proposal.

In [156], routers memorise the number of bytes of each flow passing through them. Upon reception of a packet, the number of bytes of its flow is updated and it is added to the queue so that the packets in the queue be always sorted based on the number of bytes traversed. The consequence is that flows

are given higher priority when they are at the beginning of connection. The drawbacks are that routers have flow states, heavy computations are involved (sorting the queue), and the number of the packet in the flow must be known.

[9] removes this final condition by computing the number of the packet from on the TCP sequence number. For that, it introduces a constraint: the starting sequence number must have the last $N$ bits ($N = 22$ proposed in the article) equal to 0. The drawbacks are that TCP senders have to be modified, the sequence number is more vulnerable to guessing attack, and the deployment is difficult: short flows on a standard TCP source will be penalised, since the sequence number of the first packets are misinterpreted by the router as being the $N^{th}$ packets. Two queues are used and a threshold. Upon reception of a packet, if the number of bytes of that flow is inferior to threshold, it is put in the priority queue, otherwise in the normal queue. The priority queue is FIFO, while the normal one is sorted by the number of packets traversed. The normal queue is used only when the priority queue is empty.

Another idea is presented in [42]. Only edge routers have flow states, even if not for ever. They count the packets of each flow and set the DiffServ bits of each packet. Core routers use only the DiffServ information. Edge routers mark packets as IN if the current number of packets is inferior to a certain threshold, and OUT if they exceed this threshold.

### 1.3.3 Problem formulation and motivation

The purpose of storing packets at a router queue is to absorb temporarily burst of packets. The idea we want to develop in this study is to prevent short data flows to be enqueued due to a burst induced by long-lived traffics.

To solve this problem, a possible solution is to involve only the end points. Suppose a connection which only needs to send 10 packets. If we assume the sender is aware of this small number of packets to transmit, we could propose to let him decide to send them in burst. However, this violates the slow start phase and the congestion control mechanisms principle. As a matter of fact, it results that a mechanism to favour short flows must necessarily involve the routers.

**Idea presentation** Routers are mandatory to decide if a received packet is either rejected or marked or simply enqueued. In the FIFO scheduling policy, the packet is inserted at the top of the queue.

Our packet scheduling proposal, called FavourTail, is the following. The enqueuing packet process is changed. When a packet is enqueued, a check is made in the whole queue to seek another packet from the same flow. If no other packet is found, it becomes a priority packet, otherwise it is added as normally at the top of the queue. Priority packets are added at the beginning of the queue, right after the last priority packet (if there are any). The packet reordering inside a flow is thus avoided.

This scheme is quite similar to a priority queuing scheduling mechanism [206].

FavourTail changes the packet scheduling policy; as such, it can be used with any other queue management policy such as DropTail and RED. In the simulations below we choose DropTail.

**Variants** There are many variants of this idea which do not involve flow states on routers. Some of possible variants are:

- Instead of a binary function (insert the packet in the priority queue or in the normal one), use another function $f(n, m)$ giving the level in the queue where the packet will be inserted, where $n$ is the number of packets of the same flow in the queue, and $m$ is the total number of packets in the queue. $f$ might be linear, exponential, logarithmic and so on. For example, in classical FIFO, $f(n, m) = m$, i.e. the packet is always added on top of the queue. (This is not the same as Fair Queuing, where $f$ depends on the number of packets of other flows too);

- Instead of having only two queues, it is possible to have several queues, for example one for packets with 0 other packets from the same flow, another for packets with 1-2 packets, one for 3-5 packets and the least priority queue for all the other packets. However, this solution might introduce an important overhead;

- Act on dropped/marked packets too: when a packet is to be dropped and if it is a priority packet, then choose the last normal packet to act on instead of it.

**Dimensioning** We are interested to know if FavourTail might have good results in realistic cases. As an example, suppose a flow of 8Mb/s (equivalent to 1000 pkts/s with a 1000 bytes packet size). Suppose also an RTT of 50ms. It results 50 pkts/RTT (1000 pkts/s * 50ms/RTT). If each direction has half of them, then there are 25 data packets in flight. If there are 10 routers between source and destination, then there are 2.5 pkts/router in average. We therefore consider that there are a few routers where this flow is still prioritised. However, it is more consistent to consider the overall gain obtained by the flow rather than the number of times this flow got a packet with a high priority.

**Characteristics**

- FavourTail does not only favour the beginning of connection but also flows with few packets in flight, i.e. with small congestion window;
- There is no relation between the routers, i.e. a flow with ten packets in flight, one in each router, is favoured, while another flow with only two packets in flight, both on the same router, is not favoured. This is especially true in the TCP slow start phase: packets are generated in burst, so the probability to have several packets of this flow in a router is higher. Evenly-spaced packet sending, such as in TFRC, should be much more prioritised, because their packets are distributed across all the routers;
- There are still several cases where a flow with a very small transmission time has the same transmission time in both cases (DropTail and FavourTail): (1) it might have only one intermediate router, so the chances to be in concurrence with other flows are smaller, (2) it might be because in FavourTail they are never high priority (it crosses routers which have only high priority packets), (3) it might be because in DropTail the routers are empty (so it is like they would be priority), (4) it might be because on the next router of the next router the prioritisation of the packet does not change anything;
- For priority flows, the more routers are in the path, the greater is the gain in terms of transmission time;
- In terms of security, sources cannot cheat by sending packets at a rate making them always priority, because they cannot estimate how many priority packets are inside routers (a priority packet may be put in the head if there is no other priority packet, but it may be put as number 10 is there are already 9 priority packets in the queue);
- FavourTail is NAT-compatible, if source port and destination port are used to identify flows (as they are unique for NAT too);
- The deployment is in the interest of the person who deploys. Indeed, when a router uses FavourTail, his own users are advantaged: clients receive faster Web pages, and servers reply faster to their Internet clients;
- Starvation may occur for long flows in a router which receives only priority packets. In order to remove this, the normal queue could be served from time to time even if the priority queue is not empty; we will tackle this in a future work;
- A solution which acts on two bursty packets exactly like two spaced packets, i.e. which spaces the bursty packets, would avoid TCP problem given below. We reserve this possible enhancement in a future study.

FavourTail has been implemented in NS2 as an extension of the DropTail queue. The next sections present the results.

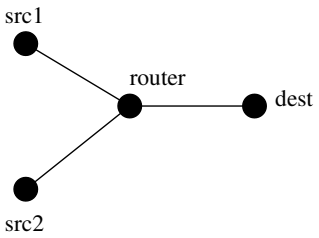



Figure 1.9: Topology of the simple network.

### 1.3.4 Simulation results

#### Simple network

We firstly evaluate FavourTail over the simple network topology given figure 1.9. The links are configured as follows: both access links have a capacity set to 2Mb/s while the bottleneck link has a 1Mb/s capacity. All links have a transfer delay set to 10ms. Two FTP/TCP traffics are generated: the first one, C1, from src1 to dest, and the second one, C2 (in the second simulation we use TFRC instead of TCP), from src2 to dest. C1 starts at sec. 0 and ends at sec. 5. C2 starts at sec. 1 and generates only 12 data packets. Obviously, at the beginning of the connection, a SYN packet is generated, and FavourTail will give it a high priority. However, it is important to keep in mind that not only the SYN packet will get a priority. The tests are made with both policies: DropTail and FavourTail.

**Results** For TCP, the time elapsed between the last packet sent and the first one for C2 is 0.53 for DropTail and 0.43 for FavourTail. The total number of packets sent by C1 is 591 in both cases. No packet is lost.

This is a positive result for the user in terms of transfer duration. Indeed, C1 is not penalised at the end of the connection, while C2 finishes about 20% sooner. The reason is that the two first packets of C2 are prioritised by the router. In fact, when arrived at the router, the first packet overtakes 13 slots, while the second one overtakes 14 slots. The difference in time for C2 ($0.53 - 0.43 = 10$ ms) corresponds to the processing time of packets by the router ($13 + 14 = 27$ packets), time gained by C2.

If C2 uses TFRC instead of TCP, the time elapsed between the last packet sent and the first one for C2 is 0.54 for DropTail and 0.17 for FavourTail. The total number of packets sent by C1 is 591 in both cases.

This is again a very positive results for the user. C1 is not penalised at the end of the connection, while C2 transfer is about 3 times faster. In fact, 6 packets from C2 are prioritised, gaining each one between 14 and 17 slots.

**Discussion** While the results with this network topology are positive, i.e. the small flow is indeed favoured, the question is why not *all* its packets are prioritised.

The explanation is the following. TCP is a protocol which sends packets in burst. The first burst contains one packet, so it is prioritised. The second burst, sent when the acknowledgement arrives, contains two packets. The first packet is prioritised on the router, but the second one arrives before the first one leaves the router (the link after the router, 10ms/1Mb, is slower than the link before the router, 10ms/2Mb), hence it is not prioritised. As the queue contains many packets from the long flow, this packet is still in the queue when the packets of the following burst arrive. In fact, because the queue has many packets, none of the following packets are prioritised.

On the other hand, TFRC is a congestion control which sends evenly-spaced packets, without burst. More packets than TCP are prioritised, but not all (6 out of 12/13). The reason is that for the seventh packet the throughput of the TFRC flow is a bit higher than the link at the right of the router can process, so it arrives at the router before the sixth packet has leaved it. The following packets suffer from the same problem.

| | DropTail | FavourTail |
|---|---|---|
| Sum of transmission times (in seconds) | 2618.68 | 2410.34 |
| Number of lost packets | 2470 | 1608 |
| Number of lost data packets | 913 | 626 |

Table 1.1: Global metrics.

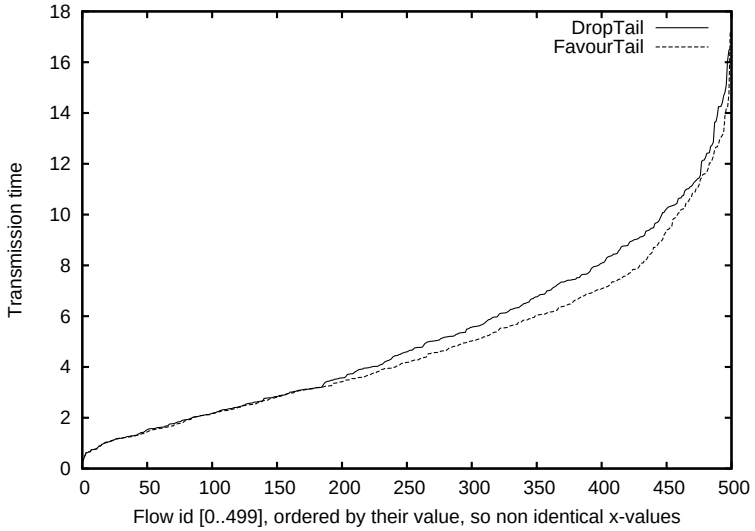



Figure 1.10: Transmission time of all the flows; the two curves have different values on abscissa.

### Complex network

In order to evaluate FavourTail over a more realistic case, a more complex network is used, the same as in figure 1.5 (page 10). Each of the five loops has 8 routers. Each router (except the 5 core routers) has 2 DSLAMs connected to it, and each DSLAM has 3 hosts connected to it. Each link has the following characteristics: 10Mb/s bandwidth, 10ms delay, DropTail (or our FavourTail).

We emit 500 FTP over TCP/Newreno connections (flows) with random and non identical hosts as source and destination. Each connection starts at a random time between 0 and 20 seconds and sends a random number between 10 and 600 packets.

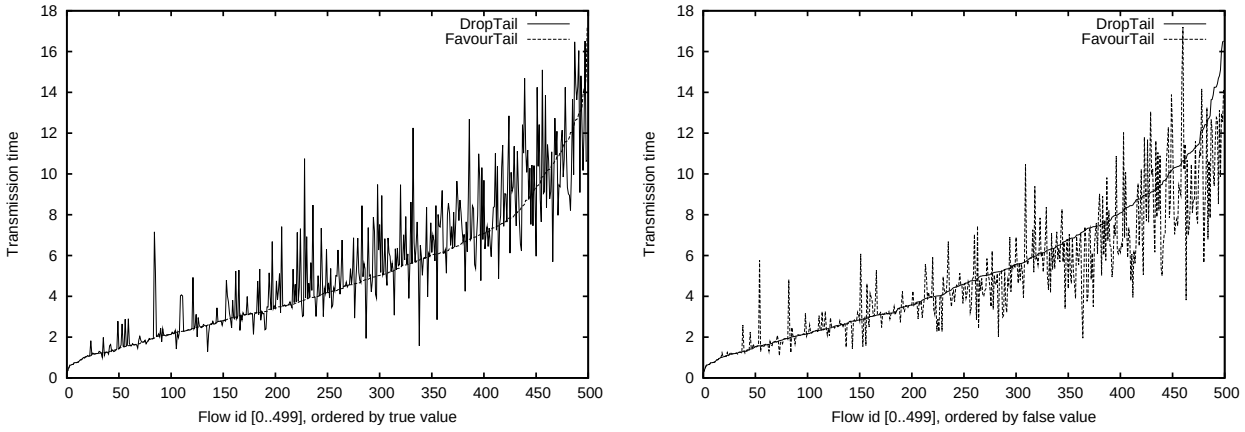
**Results** Several metrics are used to compare the classical DropTail and FavourTail, we divide them in global and short flows specific ones.

Global metrics refer to metrics about the whole simulation. Table 1.1 presents some results. It can be seen that FavourTail globally reduces the transmission time of all the flows. The time reduction is globally dispersed among the flows, as shown in figure 1.10, but among the flows with higher transmission time, as will be detailed later. As a matter of fact, if a scheduler treats first the shorter tasks, then the total finish time is smaller. This might explain the reason why we obtain a shorter transmission time. We also notice that the number of lost packets, and of lost data packets, is much smaller with FavourTail by about 30%.

A second metric is specific to short flows. Our idea favours packets when they are alone (i.e. no other packets from the same flow exist) on their current router. This leads to the idea that flows with few packets to send should be favoured. However, several results prove the contrary.

The transmission time of each flow for both variants is given in figure 1.11. They are ordered by the transmission time of the FavourTail variant, in figure 1.11(a), and the DropTail variant, in figure 1.11(b). In figure 1.11(a), it seems that flows with short transmission times are indeed favoured. However, this is not true. This is clearly shown in figure 1.11(b), where for short transmission times (on the left x-axis in this figure), the DropTail variant seems to have smaller times.

It results that using the DropTail or FavourTail transmission time is subjective. An objective comparison, i.e. order the abscissa based on the "length" of flow, is therefore needed. Several criteria may be chosen as length of flow: (1) number of packets sent by the flow, (2) number of packets




(a) FavourTail (b) DropTail

Figure 1.11: Transmission time of all the flows, ordered by the transmission time of FavourTail variant, and DropTail variant.

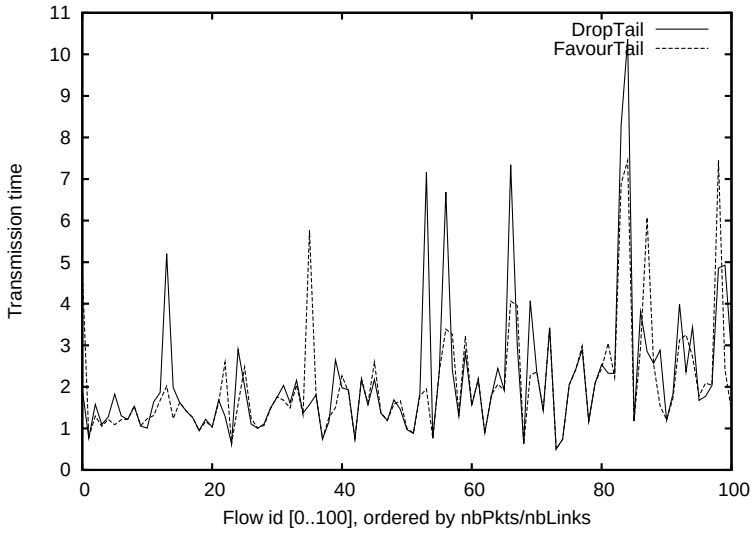



Figure 1.12: Transmission time of the first 100 flows, ordered by the number of packets divided by the number of flow links.

divided by the number of intermediate routers (or links), (3) number of packets divided by the number of concurrent flows inside path routers, and so on. For our purposes, i.e. favouring packets with few packets, the most appropriate criterion is the number of packets divided by the number of intermediate routers (we use here a simple ratio function for simplicity purpose). A smaller value means more favoured. In fact, the more the packets, the smaller the gain of FavourTail; the more the number of links, the higher the gain of FavourTail.

Figure 1.12 presents the transmission time ordered by the number of packets divided by the number of links. The curve for FavourTail is more often below the other curve, so FavourTail is slightly better. The difference between the two curves are not however high. The plot for ordering by the number of packets shows similar results (albeit the difference between both variants is a bit further reduced).

Several parameters of the simulation have been changed, the results are quite similar. The parameters changed are:

- all the traffic is TFRC instead of TCP;
- the data size of flows are not random, but use Pareto distribution;
- the first 50 flows have between 1 and 10 packets, all the others have between 200 and 800 packets to send;

- each core router has attached only one DSLAM, and each DSLAM has only one host attached.

A third metric is the influence of the router queue size. We vary the queue size of each router buffer from 2 to 200 packets. This allows to emulate a congestion level inside the whole core network. We use the flower network topology previously presented with random size of data transfer.

As expected, and in order to verify our implementation, when the queue size is very small (i.e. when the queue size is set to two), the difference is nil meaning there is no advantage for both mechanisms. When the buffers size are large, meaning that there is no congestion inside the network, our proposal does not bring any advantage. However, when the queue size is ranging from 10 to 70, that can symbolize a congestion level considered as severe to slight, our proposal allows to decrease the number of dropped packets and as a result, the number of retransmitted packets which directly results by a lower transmission time for the TCP flows.

### 1.3.5 Conclusions

This section presented and evaluated a novel forwarding scheme. We show that this scheme leads to interesting properties allowing to decrease the overall transfer delay of TCP flows in the context of best-effort networks. The results obtained are quite unexpected, as intuitively we would expect a stronger benefit of this mechanism to short TCP flows and on the contrary measurements show an overall benefit for long flows without high impact over short ones with our proposal compared to DropTail. Indeed, the main findings are that FavourTail decreases the number of packets dropped and as a primary consequence, the sum of transmission times is thus reduced. However, the short flows are not noticeably favoured compared to the other longer flows.

## 1.4 Analysis of congestion control in sensor networks

This sections studies different congestion control methods applied to a centralised control system, consisting in several sensors/actuators and one controller. Sensors/actuators are linked to the controller through an IP network. Depending on the data exchanged, the network can be congested. In such case, the congestion control used by data exchange becomes important. We evaluate four congestion control methods used by three classical transport protocols, UDP, TCP and DCCP. Simulation results on a centralised control system show that TCP and DCCP offer a good trade-off on reliability vs. throughput, whereas UDP has best results provided that the network is well configured.

### 1.4.1 Introduction

Control systems are devices which manage the behaviour of systems. Closed-loop control systems consist of sensor(s) which gather data from the environment, actuator(s) which act on the environment, and controller(s) which receive data from sensor(s) and give commands to actuator(s). In our case, we are interested in Distributed Intelligent MEMS (DiMEMS) systems [29] which comprise all these elements but at a micro-scale. The Smart Blocks[2] and the Claytronics[3] projects are good examples of DiMEMS systems.

Control systems can be as simple as a sensor, an actuator and a controller. They can be centralised, consisting of several sensors and actuators but a single controller. They are distributed if there are several controllers too.

A networked control system (NCS) is a control system where the components are connected together through a network, and which has real-time constraints. The network is shared among the components either because it allows to reduce system costs or because the interconnection is wireless. NCSs have therefore a broad range of applications such as mobile robotic systems, ground-based like in RoboCup or unmanned aerial vehicle (UAV) [39].

The network is shared among the components. Designing the network capacity so that congestion never appears is not always feasible, for example when data generated is not known in advance or

[2] `http://smartblocks.univ-fcomte.fr`
[3] `http://www.cs.cmu.edu/~claytronics`

when network has no fixed bandwidth such as wireless networks, which are subject to interference. So congestion can occur if data transmitted exceeds in size the network capacity. Congestion control is a mechanism to regulate sending rate so that data is sent at the fastest rate that still avoids congestion. This can be thought as network resource optimisation. When congestion occurs in network, data is lost. It is therefore important to analyse congestion control impact on data transmission.

In NCS research, data transfer is generally treated with low-level protocols, such as Ethernet and TDMA. Very few papers deal with congestion control in IP-based systems, yet this is a known challenge: "These challenges [for NCS] involve the optimization of performance in the face of constraints on communication bandwidth, congestion, and contention for communication resources, delay, jitter [...]" and "When communication channels in a data network are shared resources among multiple user nodes, network congestion and contention for bandwidth pose challenges for control implementations in which there are hard real-time requirements" [10]. [157] for example notices that different types of data exist in an NCS, hence different protocols or different priorities for packets can be used. In this context, [58] shows promising results for packet prioritisation, results which are even more appropriate to control systems since the network is under the control of one organisation.

Our contribution is to discuss the interest of having congestion control in NCS. We analyse simulation results of classical congestion controls used above IP-based networks.

### Background

**Congestion control** The IP protocol, used also by Internet, does no attempt to optimise network usage when sending data, because it does not take into account availability of network resources. Network optimisation has been left on/to the upper transport protocol. The classical transport protocols on Internet are TCP [152] and UDP [150], and recently DCCP was standardised too [115]. The way resource usage is taken into account is through a congestion control. Among the transport protocols presented, UDP has *no congestion control* with no data reliability, TCP has a *window-based congestion control* with 100% data reliability through retransmission (all data arrives to destination), and DCCP allows the programmer to choose a congestion control among several standardised controls, such as TCP-like [79], *similar to TCP*, and TFRC [80], an *equation-based control*, with no data reliability either.

When UDP receives a packet to be sent to network, it sends it immediately, no matter the network state. This intuitively leads to the highest sending rate, with the drawback that packets can be lost if the network is congested.

When TCP or DCCP receives a packet to be sent to network, it checks whether network is able to transport it. If yes, it is sent immediately, otherwise it is stored in a temporary buffer until network conditions allow to send it. The role of congestion control is to take care of the timings when packets are sent to the network.

TCP congestion control works as follows. At the beginning of a transmission, sender uses a slow start phase, where data sending rate increases exponentially. When data rate exceeds network bandwidth, packets are lost and sender enters to congestion avoidance phase, a network-friendly phase where data sending rate increases linearly; this is the phase used most of the time for long enough transmissions. Upon a lost packet detection, sender sends it immediately, known as fast retransmit phase, and enters a temporary fast recovery phase where it sends data at a smaller rate until it recovers from the loss, at which time it reduces by two the data rate and goes back to the congestion avoidance phase. This congestion control is used by DCCP/TCP-like too.

On the contrary, TFRC uses a sending rate equation whose parameters are filled with transmission parameters, most notable with the number of lost packets. Each RTT (Round Trip Time, the time for a packet go and back), the equation parameters are updated and data sending rate changes. This leads to a smoother changing in sending data rate, which is a useful property for some applications.

**Simulators** Control and network research communities usually use simulators to validate algorithms. Two such simulators are TrueTime and ns2, respectively.

TrueTime is based on Matlab/Simulink and targets real-time and embedded systems. The simulation model is based on three blocs:

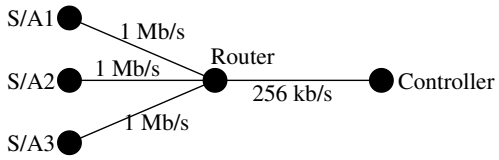



Figure 1.13: Network used for simulations.

- TrueTime kernel, for logical controls and data processing;
- TrueTime network, for packet exchange in a local network such as Ethernet, CAN, TDMA and FDMA;
- a controller.

A fourth optional block can be used, TrueTime battery. This simulator focuses on low-level simulations. Instead, we are interested in high-level protocol simulations, where congestion control is implemented.

Network simulator version 2 (NS2) is usually used in the network research community. It provides low-level and high-level protocols, and several transport protocols are supported. The accuracy of results for low-level protocols is low, but for high-level protocols, as in our case, it is high.

Given the support for various congestion controls, we choose ns2 for our simulations.

### 1.4.2 Related work

*Networked Control Systems* (NCS) lie at the intersection of control and communication theories and it is therefore relevant to compare this study to NCS ones. A NCS has four main characteristics [96, 196] that have to be taken into account in order to control the whole system, bandwidth-limited communication channels, sampling and delay, packet dropout and system architecture. If the first three characteristics match our concerns which is network transmission, the last one only deals with the architecture of the system i.e. centralisation vs decentralisation of the control. Nevertheless, studies and modelling of quantization effect [95, 139], packet drop out [193] or consensus problems [144] are important and must be taken into account together with the congestion control.

The topic of decentralised and *distributed control of systems* has been active for many years [169]. Decentralised control limits its study to special systems like spatially invariant ones [15] or nested, chained or symmetric systems [187], whereas distributed control makes less assumptions about the system [133]. Linear matrix inequalities (LMI) can be used either for decentralised [26] or for distributed systems [49]. LMI has proven to be efficient when dealing with linear systems whose physical topology is represented by an arbitrary graph [117] but these results have not been extended to non-linear systems. It has also to be noticed that these methods propose only to control large distributed system but do not help to program it. If some problems require algorithmic solutions, these methods will not be used.

### 1.4.3 Simulation results

The network used for carrying out tests is presented in figure 1.13. Three agents are connected through a router to a controller. Agents are both sensors and actuators. The sensor sampling period is 50ms; they gather information about the environment and send to the controller sensor data through the form of a packet of 1kB (1024 bytes) of application data each 50ms. Note that this gives slightly different packet sizes, since UDP, TCP and DCCP headers have different sizes. The first sensor starts sending at t=0, the second at t=100ms and the third at t=200ms. Upon reception of a sensor data (sensor packet), the controller answers by sending it an order through the form of a packet of 200 bytes. The topology is set so that there is a small congestion on router-controller link, whose bandwidth is a bit smaller than total sent data. The reason for this choice is that if a network is not congested, a congestion control brings of course no benefit, whereas we are interested to know if congestion control is useful.

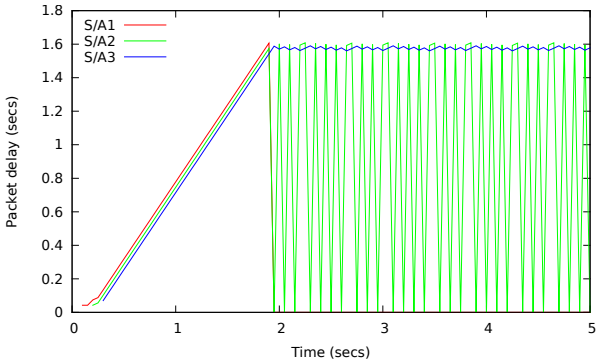



Figure 1.14: Results for UDP protocol for the first 5 seconds.

The router uses a DropTail queue management, which means that an incoming packet is rejected if and only if the buffer is full. The buffer size is set to 50 packets, the default value on the simulator. Each simulation is run for 360 seconds. The total number of packets generated by each sensor is 20 packets/sec * 360 sec = 7200 packets (in reality, between 7195 and 7199, since some sensors start a bit later).

Note that sending and receiving are done on different "cables", so they do not share the 256kb/s link. As controller answers are small, they do not provoke congestion, hence we are not interested by controller answers.

The figure accompanying each method shows the delays of each packet generated by each sensor, with the time on x-axis. A 0 delay means the packet is lost.

The layer which takes care of network congestion and is responsible for packet delays is transport layer. We tested three transport protocols, the classical TCP and UDP, and a relatively new standardised protocol, DCCP. The latter can choose its congestion control, and we tested the two currently standardised: TCP-like and TFRC.

Some applications in control systems are interested by delay (time for the sensor data to arrive to controller), others by losses. On some applications values are cumulative, i.e. the value of one parameter at one time removes the need to know the value of the parameter before (e.g. the current temperature of a system or the current position of an object), which means that losses are tolerated. On others, such as videoconferencing, the delay is an important parameter, while the reliability is important for audio and not so important for video. Therefore, we have to analyse both delays and losses in simulations.

### For UDP protocol

Figure 1.14 shows that the shapes of the delays of the three sensor data packets are very different: starting from second 2, one sensor has a constant delay of 1.5s, another one a varying delay from 0s to 1.5s, and a third one has 0 delay, meaning that all its packets are lost. What happens is a timing issue: each time a packet from the latter sensor arrives at the router, its buffer is full and the packet gets rejected. Such synchronisation is a known characteristic of a DropTail queuing discipline, as set at the router, exacerbated by the fact that UDP does not use a congestion control.

10996 packets are lost, and all of them are lost by the network. No packet is retransmitted, according to UDP specification [150]. In the figure, packets start to be lost at time 1.95s. This agrees with theoretical results: During that time, about 1.95 s * 20 packets/s * 3 sensors = 120 packets have been sent, and 1.95 s * 32 = 64 packets have been forwarded by the router. This gives a difference of 56 packets, approximately equal to the buffer size on the router, set to 50 packets, as written above.

The highest delay is 1.61s, while the average delay for received packets is 1.35s. The highest delay appears for a packet which arrives at the router while its buffer is full less one packet size, hence it takes the greatest time to be processed by the router. This explanation on the highest delay stands for the whole this section, for all the protocols below.

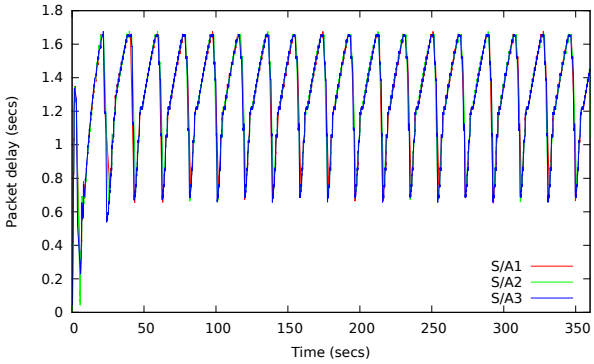



Figure 1.15: Results for TCP protocol for the whole simulation.

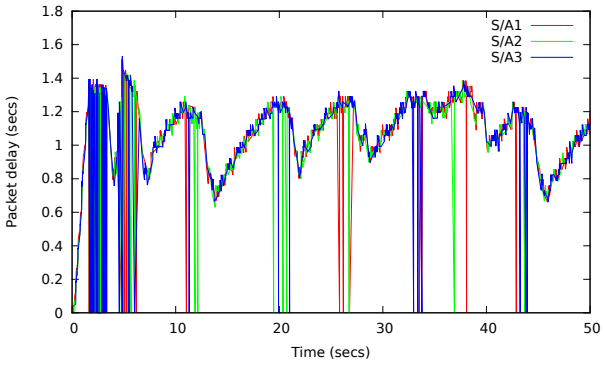



Figure 1.16: Results for DCCP/TCP-like protocol for the first 50 seconds.

### For TCP protocol

Figure 1.15 presents the results. The figure shows that the shapes of delays are similar, which is not surprising given that TCP uses a fair congestion control. This saw teeth-like curve is also classical for throughput of TCP protocol. At the beginning of the transmission, TCP increases very fast the data rate, and as a result the router buffer fills up and delay increases. Afterwards, TCP enters a loop where it increases slowly the data rate until a loss is detected, and reduces it by two when it arrives [152], which leads to similar changing in packet delay.

63 packets are lost by network, and according to TCP specification they have all been retransmitted and eventually received. 10382 packets are lost on the sensor itself, without entering the network. The explanation is that the built-in congestion control of TCP delays packets until network resources are available, and as a consequence TCP buffer fills up and eventually simply discards packets coming from the sensor.

The highest delay is 1.68s and the average is 1.38s.

### For DCCP/TCP-like protocol

Figure 1.16 presents the results. As for TCP, it shows that the shapes of the transfer time of the three sensor data packets are similar. This is not a surprising result, since TCP-like congestion control uses the same law for sending rate as TCP.

339 packets are lost by the network, which is a bit greater than for TCP. None of them has been retransmitted, since DCCP does not retransmit them (even if the exact number of each lost packets is known by the sender). 7021 packets are lost on the sensor for the same reason as for TCP.

The highest delay is 1.53s and the average is 1.09s.

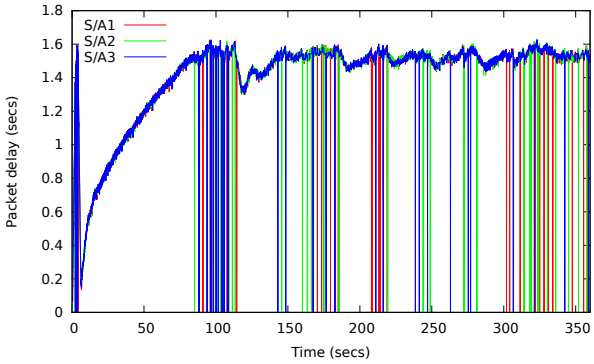



Figure 1.17: Results for DCCP/TFRC protocol for the whole simulation.

| Protocol | Sensor | Packets | | | | Delay | |
|---|---|---|---|---|---|---|---|
| | | generated | lost on sensor | lost on network | received | highest | average |
| UDP | 1 | 7199 | 0 | 7163 | 36 | 1.61 | 1.35 |
| | 2 | 7197 | 0 | 3432 | 3765 | | |
| | 3 | 7195 | 0 | 0 | 7195 | | |
| TCP | 1 | 7199 | 4425 | 0 (26 retr) | 2774 | 1.68 | 1.38 |
| | 2 | 7197 | 4380 | 0 (26 retr) | 2817 | | |
| | 3 | 7195 | 1577 | 0 (11 retr) | 5618 | | |
| DCCP/TCP-like | 1 | 7199 | 2276 | 115 | 4808 | 1.53 | 1.09 |
| | 2 | 7197 | 2510 | 108 | 4579 | | |
| | 3 | 7195 | 2235 | 124 | 4836 | | |
| DCCP/TFRC | 1 | 7199 | 3496 | 60 | 3643 | 1.63 | 1.41 |
| | 2 | 7197 | 3184 | 54 | 3959 | | |
| | 3 | 7195 | 3193 | 60 | 3942 | | |

Table 1.2: Numerical results for the four congestion controls, sampling is 50ms and simulation runs for 360 secs.

**For DCCP/TFRC protocol**

Figure 1.17 presents the results. Like TCP, TFRC is a fair congestion control, so the shapes of the transfer time of the three sensor data packets are similar.

174 packets are lost by the network, which is a bit greater than for TCP. None of them has been retransmitted, since as written above DCCP does not retransmit them. 9873 packets are lost on the sensor for the same reason as for TCP.

The highest delay is 1.63s, and the average is 1.41s.

### 1.4.4 Discussion

Table 1.2 presents a numerical overview of the simulation results (the delay columns show results for all the sensors). Several conclusions can be inferred.

Using UDP means that some sensors can be almost completely muted by synchronisation issues, as given by the first sensor. Also, UDP has by far the most losses on the network, while the protocols with congestion controls avoid it and lose almost all their packets on the sensor itself. DCCP/TCP-like has the highest number of packets received, with the three others a bit behind it.

The highest delay is more or less comparable, since it is obtained when the router buffer is filled, and this case appears for all the protocols. Note that packets can wait on sensor buffer too, but this buffer is drained much faster since the link is much greater than the bottleneck link (1 Mb/s vs. 256 kb/s). On the other hand, the average delay is very favorable to DCCP/TCP-like (1.09 secs).

These results are not surprising, as explained in the following.

First, if a network is *not* congested, a congestion control brings of course no benefit. So in the following we are interested what happens when data size is greater than network capacity.

Internet is a network where flows appear and disappear generally irregularly, because user behaviour for creating flows is unpredictable and sometimes interactive. In contrast, control systems use sensors which usually send data regularly. Such regularity sometimes appears in Internet too, for example for constant video transmission such as television broadcasting. For such cases, where data is sent at a regular pace, the sending rate smoothing done by congestion control is of no help. It should also be noted that when there are several destinations with various bandwidths, congestion control can avoid useless packet losses, situation known as network collapse, an example of which is given in [76]. This is however not the case in a centralised control system which is the focus of the current work.

The delay is affected too by the congestion control. A congestion control usually delays packet sending on sender, waiting for network availability or for acknowledgement reception (also known as TCP pacing), which increases delay on sender. On the other hand, a congestion control keeps router buffers smaller, hence reducing packet delay on routers.

UDP too has a limitation, because in conjunction with the simple DropTail queuing mechanism it leads to global synchronisation, preventing some sensors to send their data.

Given the discussion above, it turns out that UDP is not worse than TCP or DCCP, and more analysis is needed to infer what is the best congestion control in sensor networks. What is certain is that, in order to avoid synchronisation issues, UDP must be used together with a more sophisticated queuing mechanism, such as RED [78, 34], which rejects packets randomly before the queue is full.

### 1.4.5 Conclusions

This section presented a comparison among several congestion controls used by currently-used transport protocols for a centralised control system, based on simulation.

Results show that UDP in conjunction with a simple queue management mechanism has some crucial issues, but a more sophisticated mechanism should solve them. Aside that, given that the network is slightly congestion, no protocol or congestion control is definitely better than the others in terms of number of packets received. This is because in a control system data is generated at regular intervals, and the smoothing effect of congestion control has no effect. On the other hand, in terms of delay, DCCP/TCP-like gives best results. What parameter is more important depends on the control system.

## 1.5 Differentiation between wired and wireless packet losses

One major yet unsolved problem in wired-cum-wireless networks is the classification of losses, which might result from wireless temporary interferences or from network congestion. The transport protocol response to losses should be different for these two cases. If the transmission uses existing protocols like TCP, losses are always classified as congestion losses by sender, causing reduced throughput. In wired networks, ECN (Explicit Congestion Notification) [164] can be used to control the congestion through active queue management such as RED (Random Early Detection). It can also be used to solve the transport protocol misreaction over wireless networks. This section proposes a loss differentiation method (RELD), based on ECN signalling and RTT (Round Trip Time), and applied to TCP-like. TCP-like is one of the three current congestion controls present in the new transport protocol DCCP (Datagram Congestion Control Protocol). Our simulations, using a more realistic simulated loss error model for wireless networks, show that RELD optimises congestion control and therefore increases the performance of transport protocols over wireless networks, leading to an average performance gain ranging from 10% to 15%.

### 1.5.1 Introduction

Wireless networks are now widely deployed and are commonly used to access services on the Internet in spite of lower performance noticed when compared to wired networks [13, 12]. Losses in wired networks are mainly due to congestion in routers, because congestion is usually handled by dropping the received packets when the router waiting queues are full or nearly full. Hence, losses in wired

networks can be seen as an indication of congestion. This is different in wireless networks where losses often occur for various reasons, for example due to interference or poor link quality (high distance between the base station and the mobile device).

IEEE 802.11 already includes mechanisms to combat losses at the MAC layer. Wireless devices retransmit lost packets on a wireless link a certain number of times (6 for example). However, in case of long interferences, a packet can be lost 7 times consecutively on a wireless link. In this case, the device drops it and the transport level of the source discovers the loss. We are interested on loss processing at transport level.

The performance degradation reported on wireless networks appears because TCP (Transport Control Protocol) [152], commonly used by Internet applications and initially designed for wired networks, classifies any data loss as a congestion loss; therefore it reacts by reducing the transmission rate. However, in wireless networks, losses are not necessarily caused by congestion. There are many proposals on how to optimise the transport protocols performance on wireless networks in the literature; the main idea is that *transport protocols should reduce their transmission rate only in case of congestion and not if data is lost for other reasons* [12, 19, 17, 11].

Nowadays, more and more applications used over Internet, for example real-time media like audio and video streaming, can cope with a certain level of losses. If they use TCP, the high reliability may come at the price of great latency. UDP (User Datagram Protocol) [150], which does not have these drawbacks, lacks congestion avoidance support and flow control mechanisms. RTP (Real-time Transport Protocol) is an application protocol [173] widely used for streaming multimedia content (usually on the top of UDP). It allows the receiver to reorder received packets thanks to the sequence number included in the RTP packet header. RTP also uses a timestamp field which is useful in the context of real time applications synchronisation. On the other hand, RTP, like UDP, does not deal with network conditions because it also lacks a congestion control.

Another promising protocol for these applications is DCCP (Datagram Congestion Control Protocol), recently standardised as RFC4340 [115], since it does not provide reliability but allows the use of congestion control protocols. One interesting point of DCCP is the freedom of choice for congestion control protocol: TCP-like [79], which reproduces the AIMD window progression of TCP SACK, or TFRC (TCP-Friendly Rate Control) [80], among others. As described in [115], DCCP implements bidirectional and unicast connections of congestion-controlled unreliable datagrams, and also:

1. negotiation of a suitable congestion control mechanism

2. acknowledgement mechanisms for communicating packet loss and ECN (Explicit Congestion Notification) information

3. optional mechanisms that indicate to the sending application, with high reliability, which data packets reached the receiver, and whether those packets were corrupted, dropped in the receive buffer or ECN marked.

On the other hand, DCCP suffers from the same problem as TCP in wireless networks, meaning that any data loss is considered to be caused by congestion.

Because of all reasons mentioned before and because wired and wireless are often conjointly used, there is an increasing need for transport protocols to take into account the properties of wireless links and the various reasons for data loss. In this section we propose a new approach (*RELD, RTT ECN Loss Differentiation*) based on TCP-like over DCCP. It uses ECN in conjunction with RTT as the main factor to differentiate congestion losses from wireless losses, see section 1.5.5. RELD is an evolution of our previous method EcnLD [158] for loss differentiation, with an enhanced scheme that allows better realistic measurements than those obtained in classical ns2 simulator.

Contrary to some articles presented in related work (next section), and also of our previous article [158], which used simple (homogeneous) error loss models for wireless links, the new results show that in a real wireless environment where wireless losses are not uniform, *RTT increases for wireless losses, and not for congestion losses*. In fact, an interference on the wireless channel will prevent the communication to continue throughout its duration. Hence, all packets sent during the interference time are buffered in the wireless access point. For each of them, a fixed number of MAC retransmissions

is done, then the packet is dropped if it is still not acknowledged at MAC level. This buffering leads to an increasing of the RTT value for the packet which arrives just after the end of the interference. The results obtained confirm this idea.

### 1.5.2 Related work

Many approaches have been proposed in the literature to differentiate losses. They are classified into three categories.

**Intermediate agent** Certain approaches impose implementation of an intermediate agent between the source and the destination which is localised normally at the base station. Snoop [13, 12] is a TCP-aware link layer approach for local retransmission. It resides on a router or a base station and records a copy of every forwarded packet. Then, it inspects the ACK packets and carries out local retransmissions when a packet is corrupted by wireless channel errors. Other similar approaches, like ELN (Explicit Loss Notification) [11], can also be used to inform the sender that a loss has happened over wireless or wired networks. Although this kind of approach has a specific application field, it is necessary to make changes to the current base stations. Additionally, it needs more processing power at the base stations to process each packet.

**End-to-end** In this case, these approaches use end-to-end mechanisms. They do not require any network infrastructure changes. These methods can generally be classified into two main categories: those which depend on IAT (Inter Arrival Time) and those which depend on ROTT (Relative One-way Trip Time).

**Parsa and Garcia** in [147] consider losses as an indication of congestion if ROTT is increasing, and wireless losses otherwise.

**Biaz** [17] and its modified version **mBiaz** [38] use packets inter arrival time (IAT) at the receiver to classify losses. Biaz considers that when a packet arrives earlier than expected then a congestion loss has happened before. For wireless losses, the next packet arrives at around the time it should have, i.e. for $n$ lost packets, if $(n+1)\,T_{min} \leq T_i < (n+2)\,T_{min}$ then the $n$ packets are congestion losses. Otherwise, wireless losses.

mBiaz corrects an important misclassification for congestion losses. It makes a little modification to the high threshold of Biaz which becomes as follows: $(n+1)\,T_{min} \leq T_i < (n+1.25)\,T_{min}$.

**SPLD** (Statistical Packet Loss Discrimination) [146] depends also on IAT. This scheme has a packet monitoring module to collect information about arriving packets. If during a certain time there are no losses, a statistical module updates the minimum IAT and the average. Then when losses occur, a discriminator module use $IAT_{avg}$ to classify losses. SPLD considers that a loss is due to congestion if current IAT is greater than or equal to IAT stable ($IAT_{avg}$), otherwise it is a wireless loss.

**Spike**, derived from [185], is a method based on ROTT. In Spike, the packet is either in Spike state or not. A loss is considered a congestion loss in Spike state, and wireless loss otherwise. A packet enters Spike state when $ROTT > B_{spikestart}$, where $B_{spikestart}$ is the threshold indicating the maximum ROTT, and it leaves it if $ROTT > B_{spikeend}$, where $B_{spikeend}$ is the threshold indicating the minimum ROTT.

**ZigZag** [38], in addition to the deviation and the average of ROTT, is based on the number of losses $n$. If:

1. $n = 1$ and $rott_i < rott_{mean} - rott_{dev}/2)$, or
2. $n = 2$ and $rott_i < rott_{mean}$, or
3. $n = 3$ and $rott_i < rott_{mean} - rott_{dev}$, or
4. $n > 3$ and $rott_i < rott_{mean} - rott_{dev}/2$

then the $n$ losses are considered as wireless losses, and congestion otherwise.

**ZBS**, described in [38], is a hybrid algorithm using ZigZag, mBiaz and Spike which chooses one of them depending on the following network conditions:

if $(rott < (rott_{min} + 0.05 * T_{min}))$ use Spike;
else if $(Tnarr < 0.875)$ use ZigZag;
else if $(Tnarr < 1.5)$ use mBiaz;
else if $(Tnarr < 2.0)$ use ZigZag;
else use Spike
where $Tnarr = T_{avg}/T_{min}$ (the average and the minimum inter arrival time).

**TD** (Trend and Loss Density based) [45] uses the trend of the ROTT and the density of losses. Authors observe that first, congestion losses often occur around and after a peak of ROTT curve and the network congestion last for a period of time after that. Second, rare are the cases when a congestion loss happens alone. Generally, a single packet lost is regarded as a wireless loss. So, TD uses loss trend to indicate if the packet loss happens around the ROTT peak curve or not and loss density to precise how often the loss occurs.

Finally, **Barma and Matta** in [16] is another end to end algorithm but uses the variance of RTT. Contrary to our results, they notice that RTT is high for congestion losses and low for wireless losses. In our opinion their results are based on a wrong assumption in the theoretical model[4] and in the simulation model used: to simulate a wireless network, a wired link with transmission errors was used[5], however the MAC retransmissions are not simulated, which means that the RTT increasing due to MAC retransmissions is not taken into account (see section 1.5.4 for detailed information).

Performance evaluation in [27] shows that methods based on ROTT perform better than those based on IAT because losses often appear around the peak of ROTT. Methods like Biaz and mBiaz have problems when several streams share the wireless link. Spike performs better than TD under the situation of low traffic but TD is better in case of high network congestion.

**ECN** Sender uses ECN (Explicit Congestion Notification) marking. Normal utilisation of ECN to distinguish a congestion from a wireless loss works by testing the last interval of time in which a loss happened. If the source had previously received an ECN, then it indicates congestion, if not, it indicates a wireless loss. TCP-Eaglet [18] authors consider that ECN marking does not work all the time for classification losses. They propose to halve sending rate when either TCP is in Slow Start phase and there is one or more losses, or TCP sender has an ECN indication in Congestion Avoidance phase as a response to imminent congestion.

Another method, similar to TCP-Eaglet, is ECN-D [198]. According to ECN-D, two scenarios are possible:

1. there are only wireless losses in the current congestion window (cwnd)
2. wireless losses occur simultaneously with congestion losses.

For the first scenario, ECN-D finds out that a wireless loss occurred because of the non presence of ECN notification. So a loss is considered as congestion loss if and only if there is an ECN mark. Additionally, for better performance, the value of cwnd at the sender is reduced only once per window in presence of ECN marks. ECN-D is proposed to optimize the SCTP performance which does not support the use of ECN messages.

Our results show that TCP-Eaglet and ECN-D are not efficient differentiation schemes because they do not take into account congestion losses without ECN mark. However, as our RELD belongs to the same category, we evaluate in this work the performance of RELD with regard to TCP-Eaglet (same idea as ECN-D).

### 1.5.3 Simulation environment

In this work, it is mandatory to dispose of realistic lower levels models (radio propagation and medium access control), as we are interested in the impact of the real radio environment on DCCP communications and the ways to overcome the resulting problems.

[4] "The basic tenet of our approach is that if the packets are suffering congestion losses, the observed RTTs will vary but if packets are suffering random losses, the observed RTTs will not vary much".

[5] "These [wired] links represent wireless links with transmission errors".

**Propagation and loss model**

From the point of view of a higher level protocol such as DCCP, only lost packets are really taken into account (thanks to the layering of network protocols, DCCP is not designed to be aware of radio attenuation, collisions or interferences). However, the way these losses appear, their frequency and their distribution over time have a great impact on the behaviour of DCCP and the enhancements we are proposing in this paper. Different ways of simulating realistic losses exist. A first, simple, one would consist in using a traditional radio propagation model (such as the well known *tworayground* or *shadowing* models of NS2), and then add a loss or error model which drops a certain amount of packets. This is a perfectly viable solution, but we have chosen instead to use a more realistic propagation model, which will cause all the losses by itself, as it can be much more easily linked or derived from a real environment.

Many models exist, each one having its own strengths and drawbacks. Depending of the context, one has to be particularly careful when choosing the model used. Several families of models that can be found in the literature: Friis [81], models integrating a kind of *fast-fading* causing packet drops, e.g. Gilbert-Elliot model, *shadowing* [199], more complex models but computationally-intensive [179], even more advanced propagation algorithms [53], or take a different approach and essentially use real collected data [70]. The simpler models (*Friss, tworayground, shadowing, ...*) are not detailed nor realistic enough for us to use here, in particular because of their simplistic losses distributions over time. On the other and, models which require an extremely detailed modelling of the environment (for use with raytracing or similar techniques) produce results that are only valid in the specific context that was modeled.

For these reasons, we decide to use the *shadowing-pattern* model described in [62] and [63], which is based on the standard *shadowing* but adds the ability to change the signal strength in a bursty way, which in turn produces statistically realistic bursty losses. The *shadowing-pattern* model was originally designed for use in vehicular ad hoc networks, but is quite versatile. This model mimics the reality where different elements in the environment have cumulative effects on the strength of the signal at the receiver. It makes use of so-called *perturbators*, which are an abstraction of real-world elements (or group of) that have an impact on the signal strength. Those elements can be as varied as peoples or cars passing by, doors opening and closing, other nodes sending data over the radio channel, etc. *Perturbators* affect a limited and configurable area of the virtual environment. They alternate between two states, *active* and *inactive*. In *active* state, they affect, usually by reducing the strength, any signal received by a node inside the area of effect of the *perturbator*; in *inactive* state, they have no effect at all. Each perturbator is thus defined by the time and the standard deviation of the time spent in both states, along with its strength when active. Figure 1.18 shows how the individual effects of two perturbators evolve over time and also how they can be combined and sometimes pass a threshold they would not have been able to if taken individually.

As explained in [62], this techniques does not aim for an accurate modeling of a particular environment. It instead aims at producing extremely realistic statistical behaviours, particularly in term of losses distribution over time. It is also easy to configure and requires lightweight computations. In fact, when using *shadowing-pattern*, one has just to choose a set of perturbator and configure their parameters. As their effects combine, a set of a few (2 to 5) perturbators is usually enough to describe a quite realistic environment. And as any average and standard deviation can be given, it can be used to model real world phenomena of any time scale.

**Simulation topology**

Figure 1.19 shows the dumbbell topology employed for carrying out the simulations. It is a wired-cum-wireless topology with 2 senders (s1 and s2) and 2 receivers (m1 and d1). The link between routers R1 and R2 is the bottleneck for wired network. The wireless network uses the standard 802.11 for wireless communications, with a bandwidth of 54Mbit/s. Each node (routers, access point and edge nodes) uses RED for queue management with default values. ECN is enabled on all of them. The packet size is 500 bytes and the simulation time is 60 seconds.

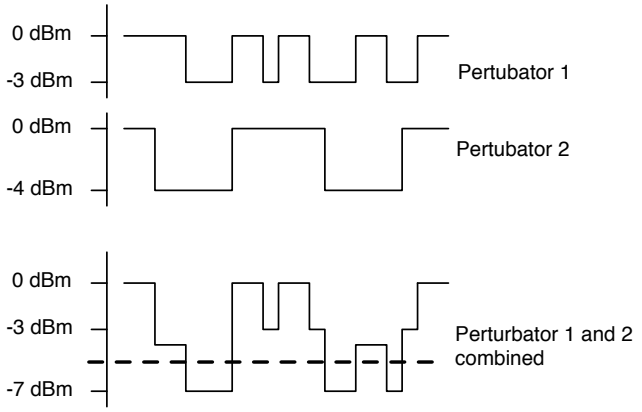



Figure 1.18: Effects of perturbators over timer and combination

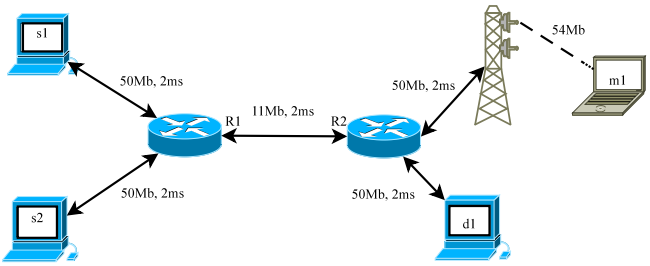



Figure 1.19: NS2, network topology.

### Configuring the physical network

The simulator used is NS2 [140] version 2.34. In the following, in order to evaluate our propositions in a range of realistic contexts, we will run 51 different tests over this topology, changing the parameters of the radio propagation model.

The difference between the 51 tests is the level of wireless perturbation added to the wireless network. Perturbations on the wireless channel are performed using *shadowing-pattern* perturbators. A mix number of zero to three out of seven perturbators is used in each of these tests. Table 1.3 shows these seven perturbators, their power, when they are active and when they are not. As the simulations presented are focused on standard WiFi networks (a mobile computer accessing the network through a base-station), and as we are interested in DCCP flows, we will only use relatively high frequency *perturbators*, with individual effects lasting no more than one hundredth second. It is obvious that phenomena that could prevent any communications in the network for many seconds or even hours are beyond the scope of DCCP flow control optimisation. Also keep in mind that even no individual *perturbator* effect lasts for more than a few hundredths of a second, multiple perturbator effects and durations can overlap. In such (common) event, they effectively prevent communications for a longer period.

Because of the chosen fixed topology, all perturbators taken alone except number 7 had no effect on the reception threshold of wireless channel. When two of them are combined and active the signal attenuation brings the wireless signal under the reception threshold which translates to packet loss on wireless channel. Perturbators number 1 and 2 have the same power but with different time effect (2 is stronger than 1). Same thing for 3 and 4 (4 is stronger than 3). So, in total we have one test without perturbation and 50 tests with a number of one to three mix perturbators. This variety of tests aims at producing a group of realistic wireless environments, ranging from absolutely not to very disturbed radio channels.

| *perturbator number* | *power* (dBm) | *inactive time* (sec.) | *standard deviation* (sec.) | *active time* (sec.) | *standard deviation* (sec.) |
|---|---|---|---|---|---|
| 1 | -2 | 0.03 | 0.005 | 0.01 | 0.01 |
| 2 | -2 | 0.01 | 0.02 | 0.03 | 0.01 |
| 3 | -3 | 0.03 | 0.005 | 0.01 | 0.01 |
| 4 | -3 | 0.01 | 0.02 | 0.03 | 0.01 |
| 5 | -4 | 0.03 | 0.01 | 0.02 | 0.01 |
| 6 | -5 | 0.04 | 0.01 | 0.02 | 0.01 |
| 7 | -6 | 0.04 | 0.01 | 0.01 | 0.01 |

Table 1.3: The seven perturbators.

### 1.5.4 Influence of losses on RTT

Losses are traditionally caused by buffer filling on routers. However, nowadays wireless networks are ubiquitous, and in these networks losses are numerous and increase with signal degradation according to various environment circumstances: distance between mobile and base station, high bit error rate related to wireless link and so on. Such losses induce sender to halve its sending rate, because it wrongly considers losses as a sign of congestion, which is not the case. For better performance the sender should be able to distinguish congestion from wireless losses. The purpose of this section is to show that the RTT can be used to differentiate losses.

But before this, the next section analyses the usefulness of loss differentiation.

Note that the RTT used throughout this section is the *RTT of the packet following the lost packet* (time between ack reception and corresponding data packet sending), which gives information about the lost packet.

#### Mathematical study on usefulness of loss differentiation

In this section a mathematical study is presented to analyse the effect of loss classification on the overall throughput. The model shown here is derived from [112, 16] and adapted to AIMD (additive increase multiplicative decrease) algorithm of TCP-like. In TCP-like congestion control, the ACK ratio (denoted here by $R$), corresponding to the frequency of ACKs for received packets, is defined as a parameter[6].

On each received ACK, the congestion window (cwnd) is increased by $R/cwnd$ when $cwnd \geq ssthresh$. This means that the congestion window is increased by one packet for every window of data acknowledged without lost or marked packets. On the other hand if the ACK reports lost or marked packets, cwnd is divided by 2 ($cwnd = cwnd/2$).

Suppose that $p$ is the packet drop probability for which cwnd is halved (which usually is the drop probability on overall wired and wireless networks), and RTT is the round trip time. Then the expected change of cwnd on each received ACK will be:

$$E[\Delta cwnd] = \frac{(1-p) * R}{cwnd} - \frac{cwnd * p}{2} \tag{1.2}$$

In case of one ACK each $R$ received packets, the time between each two updates of cwnd is $\frac{R*RTT}{cwnd}$. So, the rate change $x(t)$ in this laps of time is:

$$\frac{dx(t)}{dt} = \frac{\left(\frac{(1-p)*R}{cwnd} - \frac{cwnd*p}{2}\right) RTT}{\frac{R*RTT}{cwnd}} \tag{1.3}$$

This differential equation can be written like this:

$$\frac{dx(t)}{dt} = \frac{1-p}{RTT^2} - \frac{p}{2R}x^2(t) \tag{1.4}$$

[6]In DCCP, $R$ can change during a communication. In TCP, $R$ is fixed and equals 1 or 2.

Let $a = \frac{1-p}{RTT^2}$ and $b = \frac{p}{2R}$. Then by integration we have:

$$\int_0^{x(t)} \frac{1}{a - bx^2(t)}\, dx(t) = \int_0^t dt \tag{1.5}$$

which gives:

$$\frac{\tanh^{-1}\left(\sqrt{\frac{b}{a}}x(t)\right)}{\sqrt{ab}} = t + c \tag{1.6}$$

$$\frac{\ln\left(1 + \sqrt{\frac{b}{a}}x(t)\right) - \ln\left(1 - \sqrt{\frac{b}{a}}x(t)\right)}{2\sqrt{ab}} = t + c \tag{1.7}$$

$$x(t) = \sqrt{\frac{a}{b}} * \frac{e^{2t\sqrt{ab}+C} - 1}{e^{2t\sqrt{ab}+C} + 1} \tag{1.8}$$

The steady state throughput of TCP-like is given by:

$$x = \lim_{t\to\infty} x(t) = \sqrt{\frac{a}{b}} \tag{1.9}$$

By replacing $a$ and $b$ with their values we obtain the influence of $p$ on the throughput:

$$x = \frac{1}{RTT}\sqrt{\frac{2R(1-p)}{p}} = \frac{1}{RTT}\sqrt{2R\left(\frac{1}{p} - 1\right)} \tag{1.10}$$

We now compare equation 1.10 in case of loss classification and without classification. Let $p_c$ be the probability that a packet is dropped because of congestion, and $p_w$ the probability that a packet is dropped on the wireless link. In the case where wireless losses do not halve the congestion window cwnd (loss differentiation is used), $p = p_c$. In the classical case, no differentiation is used, so $p = p_c + p_w$. Let's denote also the throughput of the classical method by $x_c$ and the throughput of the loss differentiation method by $x_l$. Then, the expected gain of the throughput is:

$$\frac{x_l}{x_c} = \frac{\frac{1}{RTT}\sqrt{2R\left(\frac{1}{p_c} - 1\right)}}{\frac{1}{RTT}\sqrt{2R\left(\frac{1}{p_c+p_w} - 1\right)}} = \sqrt{\frac{\frac{1}{p_c} - 1}{\frac{1}{p_c+p_w} - 1}} \tag{1.11}$$

From equation 1.11 we can conclude that:

1. if $p_w = 0$ then $x_l = x_c$
2. if $p_w > 0$ then $x_l > x_c$ and the ratio increases while $p_w$ increases.

Otherwise said, the loss differentiation usefulness increases while the number of wireless losses $p_w$ increases.

### The impact of loss type on the RTT in theory

**Impact of a congestion loss on RTT:** Let $s$ be the time needed for a router to process and send a packet, i.e. the service time of the queue. Suppose a packet is enqueued in a router queue (see figure 1.20). Let $n$ be the place in the queue in the case the previous packet has been enqueued. If, on the contrary, the previous packet has been dropped, then the packet is placed at position $n-1$, hence it takes $s$ less time to be processed by the router. Otherwise said, after a packet drop, this router reduces the RTT of the packet by a time equal to $s$.

An example of value for $s$ is given in the following. Suppose a router with a 100Mb/s interface, and 1000 bytes packets. The interface sends at 100Mb/s = 12.8MB/s = 12.8kpkt/s. This means that a packet takes 1/12.8ms, so $s \approx 0.1$ms. For higher speed interfaces, $s$ is smaller. Figure 1.21 shows similar results, with differences of less than 1ms generally.

To conclude, the RTT of a packet *decreases* after a congestion loss.

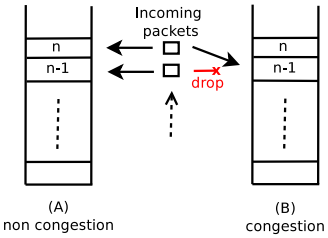



Figure 1.20: Theoretical impact of congestion losses on RTT.

**Impact of a wireless loss on RTT:** When a loss occurs in a wireless network, it is retransmitted at MAC level until either it is received, or the retry limit is reached. If retry limit is reached, the packet is simply dropped.

In order to reduce wireless network collision, for each MAC retransmission the wireless card waits, according to the standard, a certain number of slots (called a *backoff*). A slot $s$ equals $20\mu s$, and the number is taken randomly in the interval between 0 and the value of contention window (CW). CW is initialized to $2^5 - 1$ for the first attempt. If the card does not receive an ACK for the sent packet, CW doubles (without however exceeding 1023) and the transmission is tried again. This is done for each retransmission. So for the first retransmission CW=$2^6 - 1$, for the second retransmission CW=$2^7 - 1$, and generally for the n-th retransmission, CW=$\max(2^{5+n} - 1, 1023)$.

In a real world environment, and especially if the receiver is close to the range limit, the channel conditions can be bad enough to prevent multiple successive transmission attempts. If all MAC retransmissions fail, the packet is lost. Consequently, for a packet lost on the wireless network[7], the backoff contributes with an additional time of $s \cdot \sum_{i=1}^{i=n} \max(rand(2^{5+i-1} - 1), 1023)$. This time is added to the RTT of the next packet to arrive, which has waited in the queue. In case of a retry limit equal to 7 (this value is used in real networks and also in NS2 network simulator, and means 1 transmission and 6 retransmission at maximum) the additional time is equal in average to:
$20\mu s \times (31 + 63 + 127 + 255 + 511 + 1023 + 1023)/2 = 30.33$ms.

As such, the RTT increases by about 30ms for a wireless loss[8]. Figure 1.22 shows similar results, for example at second 30 the difference between average RTT and the RTT after a wireless loss is 15ms and at second 32 the same difference is 35ms.

To conclude, the RTT of a packet *increases* after a wireless loss.

### The impact of loss type on the RTT in simulation

To evaluate the impact of loss type (congestion or wireless) on the RTT in simulation, we use the network and the *shadowing-pattern* propagation model presented before. We present in this section the result of two tests, with a strong perturbator and without any perturbator, both of them using the original TCP-like congestion control under DCCP in NS2.

Figure 1.21 presents the RTT evolution in presence of congestion losses; in this figure no perturbator is used, hence no wireless losses are present. It can be noticed that the RTT is generally stable (generally between 0.034 and 0.04 seconds). Also, most of congestion losses have an RTT smaller than the average, as can be seen at 37, 38 and 39 seconds for example.

Figure 1.22 presents the RTT evolution in presence of congestion losses and wireless losses; in this figure a perturbator (number 7) for wireless channel is used. It can be seen that the wireless perturbation makes the RTT unstable (generally between 0.025 to 0.07 seconds). Second, most of congestion losses appear when RTT is below average i.e. between 53 and 57 seconds, while most of wireless losses appear when RTT is above average i.e. at 20, 22 and 30 seconds.

The previous two results, given by theory and simulation, lead to the same conclusion:

[7]Compared to the smallest transmission time (a packet which succeeded at the first transmission and with backoff equal to 0).

[8]This is the backoff contribution only; other factors, less important in our study, can modify this value, such as the transmission itself or other transmissions during backoff waiting.

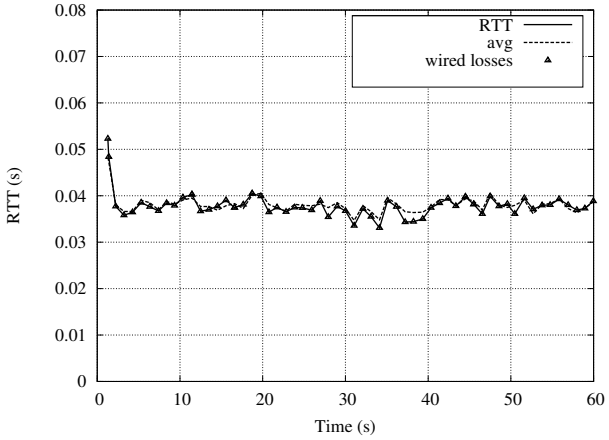



Figure 1.21: Impact of congestion losses on RTT.

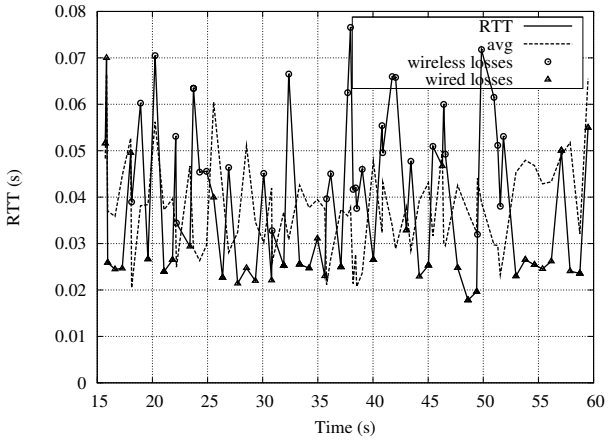



Figure 1.22: Impact of congestion and wireless losses on RTT.

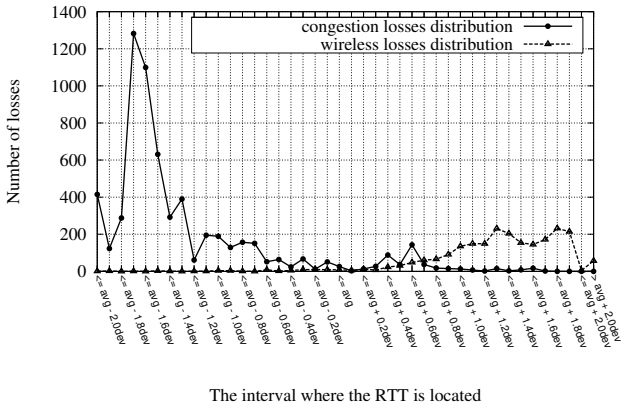



Figure 1.23: Distribution of losses based on RTT intervals.

- RTT for congestion losses are generally *smaller* than average RTT
- RTT for wireless losses are generally *greater* than average RTT

Moreover, it appears that the difference of RTT is usually greater for wireless losses than for congestion losses.

#### The choice of RTT threshold to distinguish losses

The previous section showed that it is possible to classify losses using their RTT values. This section further analyses the RTT evolution for congestion and wireless losses with respect to average RTT and its deviation. It aims to find out a threshold between congestion and wireless losses.

The results are presented as an average of the results of all the 51 tests depicted previously. Figure 1.23 presents the distribution of congestion and wireless losses around the average RTT ($avg$) using a step of one tenth of the RTT deviation ($dev$). First, this figure confirms that congestion losses have generally RTTs smaller than $avg$, and wireless losses have generally RTTs higher than $avg$. Moreover, most of congestion losses appear between $avg - 1.8dev$ and $avg - 1.3dev$, and wireless losses appear between $avg + dev$ and $avg + 1.9dev$. Using some calculus (figures detailing this are found in [163]), it can be noticed that for the 51 tests about 90% of the congestion losses have an RTT $\leq avg$. It is about 98% at $avg + 0.6dev$, and all of them have an RTT $\leq avg + 1.5dev$. It can also be seen that 3% of wireless losses have an RTT $\leq avg$. The percentage is 8% for RTT $\leq avg + 0.6dev$ and more than 60% for RTT $\leq avg + 1.5dev$. Finally, it is much more frequent to have congestion losses with RTT $\geq avg$ than wireless losses with RTT $\leq avg$.

These results show that perfectly classifying all losses using RTT is impossible. However, choosing a threshold between $avg$ and $avg + 0.6dev$ can correctly classify the majority of congestion losses and misclassify only a few wireless losses (between 3% to 8%).

We have tested a threshold of $avg$ and of $avg + 0.6dev$. The results were similar, albeit a bit worse for $avg$. As the value of 0.6 is chosen somewhat empirically, we cannot advocate that it is the *best* value for all the cases. Nevertheless, we claim that the best threshold should be a bit greater than $avg$. In the following sections we will use a threshold of $avg + 0.6dev$.

### 1.5.5 RELD, RTT ECN Loss Differentiation

The purpose of RELD is to use ECN in conjunction with RTT to prevent network congestion and to maintain sending rate in case of wireless losses. To differentiate between congestion losses and wireless channel losses, RELD requires that intermediary routers between sender and receiver be ECN compatible. For this, it is necessary that the routers implement an active queue management such as RED (Random Early Detection).

See the beginning of this chapter for information about RED and ECN.

#### RELD details

Like our method, TCP-Eaglet [18] uses ECN information. However, it does not deal with losses in Slow Start phase (the first phase of a TCP connection, where the congestion window increases exponentially with the RTT) and hence it does not consider the case where a burst of packets arrive to a router and exceed its queue capacity. In this case, there may be a significant number of ECN unmarked losses, which might appear even in Congestion Avoidance phase (the long-running phase of a TCP connection, where the congestion window increases linearly with the RTT) if other concurrent flows are in Slow Start phase.

Our contribution is that, unlike TCP-Eaglet, RELD takes these situations into account. First, it makes no difference between Slow Start and Congestion Avoidance phase. Then, contrarily to TCP-Eaglet, it uses the RTT to remedy the ECN weakness, as shown below.

As ECN marking occurs often before congestion, a responsive sender to ECN can use this information to prevent congestion and to differentiate congestion losses from wireless losses. A sender which reduces its sending rate in response to ECN can avoid congestion in most cases but not all. In fact, when a burst of packets arrives to the router, its queue might become full. Since the queue average has not changed much, the router drops packets without marking them. In such cases, losses are numerous and they often causes an RTT growth.

To sum up, RELD considers that a loss is due to congestion if and only if:

1. $ecn > 0$
   or
2. $n > 0$ and RTT $< avg + 0.6dev$

where $ecn$ is the number of packets marked EC (Experienced Congestion), $n$ the number of lost packets indicated by the received Ack, RTT the current RTT, $avg$ the average RTT and $dev$ the RTT deviation.

In this manner, RELD works as TCP-like in increasing phases, i.e. in Slow start and in Congestion Avoidance phases the congestion window will increase as usually. On the other side, when the sender receives a loss indication it will decrease its congestion window only if the losses are considered by RELD as a congestion (as described on the above formula). Formula 2 has been deduced from the results of the previous section. The complexity of formula 2 is constant, like the other loss differentiation algorithms in related work, because the calculations of RTT, $avg$ and $dev$ are made by a simple equation using constant values and just two saved values each time.

### 1.5.6 Simulation results

We have done extensive simulations [163] to analyse RELD threshold and its classification rate, measure the performance gain of RELD compared to original TCP-like, and compare RELD to TCP-Eaglet. For this, two scenarios are used: without competition and with competition of another flow.

To have a working environment, we fixed a bug in NS2 so that RED and ECN can be used on wireless links. For DCCP protocol in NS2, we used the patch written by Mattson [135] and added to it support for wireless links[9].

#### Validation of the RTT threshold chosen by RELD

In section 1.5.4 an RTT threshold has been chosen from statistical results so that it allow to distinguish efficiently between congestion and wireless losses. The congestion control protocol used for that was TCP-Like. The goal of the current section is to validate this choice by using RELD as congestion control protocol under DCCP and verifying that this new algorithm does not alter the classification.

It should be noted that in the following figures, differentiation is done by a thorough parsing of the log files. An indeed strong point of using a simulator is that it can precisely tell us where and why packets are really lost.

[9]The new patch has been found for a long time at `http://eugen.dedu.free.fr/ns2`, but was removed when it got integrated in the NS2 simulator itself. A few researchers contacted me for the patch during that time.

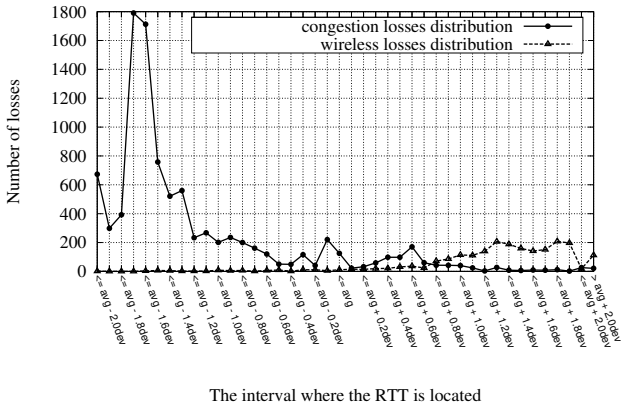



Figure 1.24: RELD, without competition: distribution of losses based on RTT intervals.

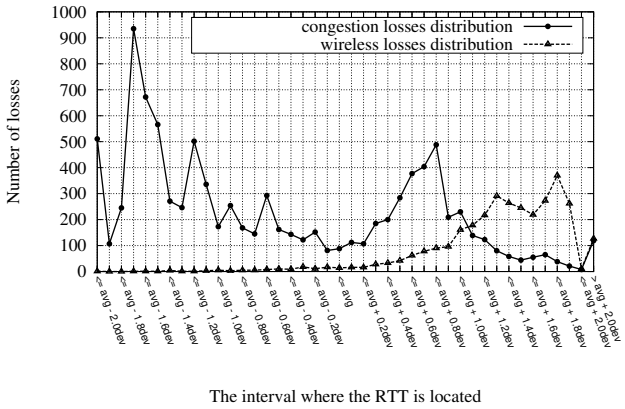



Figure 1.25: RELD, in competition: distribution of losses based on RTT intervals.

**Scenario without competition** Figure 1.24 confirms the two conclusions from section 1.5.4: congestion losses have an RTT smaller than average RTT, while wireless losses are generally above average RTT. Some calculus consolidate the choice of $avg + 0.6dev$: only 2% of congestion losses are not included in RELD formula (have RTT $\geq avg+0.6dev$), and about 10% of wireless losses are not included in RELD formula.

**Scenario with competition** A concurrent TCP flow is added to the network (figure 1.19), between s2 as a sender and d1 as a receiver, and it appears twice: from 1 to 20 seconds and from 25 seconds to 45. Its goal is to create traffic in Slow Start mode (when the queues are likely to become full, and without ECN notification) several times during the simulation.

Only the results of the 51 tests for the RELD flows are presented in the three figures. Figure 1.25 presents the distribution of losses. First, it shows that fewer congestion losses are gathered to the left margin of the graphic; in fact, congestion losses between $avg - 1.8dev$ and $avg - 1.6dev$ in figure 1.25 are twice fewer than those of the same interval in figure 1.24 (900 compared to 1800). Second, congestion losses are more evenly distributed. Third, congestion losses span more to the right, which means that the loss classification is more difficult. And fourth, as before, most of congestion losses are at the left and wireless losses are at the right. After doing some calculus, fewer congestion losses (70%) are included in the RELD formula (RTT $\leq avg+0.6dev$). Of course, choosing a greater threshold can reduce congestion losses misclassification, however this will result in a higher wireless misclassification rate. As for wireless losses, the same RTT distribution as before is noticed, which is normal because the same perturbators are used.

As a conclusion, RELD threshold allows to classify congestion losses correctly between 80% (in

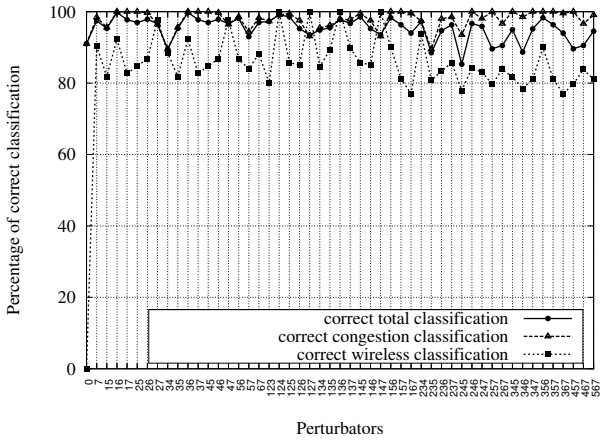



Figure 1.26: RELD, without competition: correct classification rate.

case of competition with other traffic) and 98% (in case without competition).

### Evaluation of RELD classification accuracy

In this section we evaluate RELD performance through its ability to classify congestion and wireless losses using the formula which combine ECN and RTT. This performance is evaluated for a wide rage of channel conditions; from absolutely not to very disturbed. Those conditions are expressed by the identifiers of the perturbators used for a given simulation. "0" means only perturbator 0 was used, "234" means perturbators 2, 3 and 4 were used in conjunction, etc.

The classification makes use of several parameters, explained in the following. Suppose that during a test the following results are obtained: there were 80 congestion losses and 20 wireless losses. Among the congestion losses, 70 of them are correctly identified as congestion losses, and 10 wrongly identified as wireless losses. For wireless losses, 5 are wrongly classified and 15 correctly classified. This is resumed in the following table (c means congestion losses, w means wireless losses):

| real | 80c | 20w |
|---|---|---|
| classified | 70c 10w | 5c 15w |

The percentage of correct congestion loss classification is 70/80, and of correct wireless loss classification is 15/20. The percentage of total correct classification is the total number of correctly classified losses divided by the total number of losses: $(70 + 15)/(80 + 20)$.

In the following figures, the real reason for each packet loss (congestion or wireless loss) is taken from the simulation trace file, and the considered reason for loss is printed by the source of the flow, which uses RELD classification, from the NS2 source code.

For the scenario without competition, figure 1.26 shows three curves: first one presents percentage of correct classification for total lost packets including congestion and wireless losses, second and third show correct classification for congestion and wireless separately. The total correct classification varies between 85% and about 99% in most cases, and it is about 92% in average. Correct congestion classification in this scenario without competition is very high thanks to RELD threshold which covers majority of congestion losses. On the other hand, correct wireless classification rate, while smaller than congestion one, is high too, varying between 78% and 100%. The number of congestion losses is very small (up to 600 losses, compared to between 15000 and 115000 sent packets, depending on the test), which means that the ratio of received to sent packets is very high. In other words this high ratio is very important for applications with special needs, such as multimedia streaming, since it avoids the need of packet retransmission methods; moreover, a high number of lost packets leads to quality degradation.

For the scenario with competition, as previously, a TCP concurrent flow is added in the network, between s2 and d1. In all tests the rate of correct classification is again very high, around 80% in average for total and for congestion losses. As a side note, contrary to scenario without competition, correct classification rate of wireless losses is higher than similar rate for congestion losses.

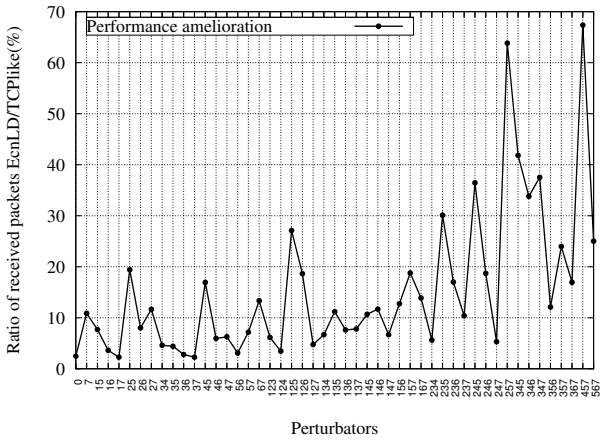



Figure 1.27: RELD vs TCP-like, in competition: performance amelioration.

The percentage of congestion and wireless lost packets compared to sent packets is very small again, and the number of congestion losses is greater than wireless losses, due to the bandwidth sharing with another traffic.

### RELD vs TCP-like

A loss classification method is supposed to make transport protocol perform better in wireless environment. To verify the performance amelioration brought to DCCP via RELD loss classification algorithm we compare the number of received packets by receiver when using TCP-like and RELD. Results are expressed in ratio between the number of received packets by the receiver of RELD and the same number for receiver of TCP-like. The ratio value indicates the amelioration, equality or the degradation of TCP-like performance.

The conclusions for the scenario without competition are identical to the scenario with competition, so we will present only the latter one.

In the scenario with competition, as previously, a TCP concurrent flow is added in the network, between s2 and d1. Figure 1.27 shows the results of the 51 tests. First, it shows that, except a very few cases, the ratio received/sent packets is greater for RELD than for TCP-like. Second, the amelioration is high, with an average of 15% and a maximum of 68%.

Summing up, RELD ameliorates the performance of transport protocol both without and with competition. Performance gain is higher when wireless losses are higher, and is smaller when congestion losses are higher.

### RELD vs TCP-Eaglet

As mentioned in related work, ECN category, there is also TCP-Eaglet which uses ECN to distinguish between congestion losses and wireless ones. It is based on a very strong assumption: since ECN is able to prevent congestion in wired networks, it is also able to distinguish losses. For it, a lost packet in Congestion Avoidance mode means wireless loss, while an ECN marked packet means congestion. Our results show that this hypothesis is often not valid, especially in a perturbed wireless environment, and as such can lead to congestion in the network.

Original TCP-Eaglet was designed for TCP, not for DCCP. However, we are interested in comparing our approach with its idea, so we implemented it for DCCP in NS2. In the scenario without competition, the correct classification rate of TCP-Eaglet, shown in figure 1.28, is very small, about 4.4% in average, most of the losses being classified as wireless. This is already a very low result, so we do not test its performance in the scenario with competition.

TCP-Eaglet indicates that its parameters must be adjusted to fit their assumption, which could be one of the reasons of the bad results of TCP-Eaglet. A second reason is that, in wired networks, an ECN marked packet has a small chance to be lost afterwards. On the contrary, this can arrive in a

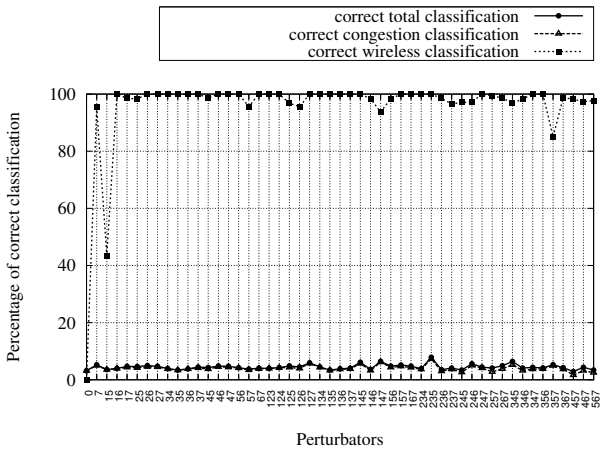



Figure 1.28: Eaglet, without competition: correct classification rate.

wireless network, known to have sometimes a certain number of consecutive losses. RELD copes with this problem thanks to the other indicator (the RTT). In this case, when marked packets are lost on a wireless channel, the RTT will increase, and RELD acts like TCP-Eaglet. However, for the latter losses, as they are congestion losses, their RTT decreases, and RELD considers them as congestion losses, hence it decreases its sending rate.

This shows that using ECN alone to distinguish losses is not satisfactory. Moreover, RELD sender reacts to congestion faster than TCP-Eaglet.

### 1.5.7 Conclusions

This section presented a general method to solve some issues related to low performance of transport protocols in wireless networks. It showed that congestion control is more efficient on wireless networks if the loss classification is correctly made between losses due to wireless media and losses due to congestion. We have also shown that ECN can successfully be used with RTT to differentiate congestion losses from wireless losses. Moreover, our statistical results done by simulations, which use a more realistic loss error model for wireless networks, confirm that RTT increases in case of wireless losses. This confirmation is contrary to some studies about loss differentiation. We recommend the use of RELD for video streaming over wireless networks because the reception rate obtained by RELD is very high (the majority of packets are received).

## 1.6 Conclusions

Congestion control is an old research field (TCP [152] appeared in 1980) which still evolves and tries to adapt to various technologies such as wireless networks and various applications such as small HTTP flows. There is a numerous literature spanned in various sub-fields, such as optimisation for long fat networks (i.e. with big BDP, bandwidth-delay product), for wireless networks, for satellite networks, for video streaming, improvement of flow starting (e.g. starting with 10 packets instead of 2–4) and many others. My contributions in this field dealt with both network core (packet loss optimisation in routers) and extremities (loss differentiation), on general computer networks (Internet) and specific ones (sensor networks). It showed that despite maturity of the field, there is room to improve it. Results are satisfactory, for example differentiation rate between congestion and wireless losses is between 80% and 100%, cf. figure 1.26.

# Chapter 2

# Adaptive video streaming with congestion control

## 2.1 Introduction

As already written in the introduction of the previous chapter, my research team worked on video transmission when I joined the team. Research on video spanned on video encoding/decoding, video servers and their databases, video analysis, transmission on network and others. The team worked on video transmission, so I started to work on this too.

Video transmission is an important research topic. In recent years, the number of videos encoded in several bitrates has significantly increased to the point that they become accessible to everybody. Videoconference usage is also increasing, and the quality of video used during videoconference sessions can vary greatly depending on the encoding quality chosen by the sender. These videos are delivered to final users using streaming services. Multimedia streaming services over Internet, as well as the demand for higher quality from final clients are in constant progression. New video standards like HD and 3D [47] are asking for more bandwidth. On the other hand, smartphones become ubiquitous nowadays but most of the videos currently available do not the match restrictive capabilities of mobile devices. According to the last Sandvine report [170], in North America, the video streaming in fixed internet connections uses about 50% of the downstream bandwidth, and similar results are also found in other continents. Video is thus the first usage in Internet, and it is very important to make the best of its use.

Video transmission can be done simply as a classical transfer, and once finished, the user can visualise it. This forces the user to wait the complete download, which can be a long time. The other possibility is video streaming, where the user starts to watch the video soon after starting downloading, during download process itself. Problems arise when the network does not have sufficient capacity to continuously provide new data to the user. Numerous works address this. Another research field is when the video transmitted or available is in a format not suitable to the user, for example it has a resolution higher than the user's smartphone. Several video flows can also be mixed, for example in a videoconferencing with several distinct persons. In this case, the work of mixing/transcoding can be done either by server or by a component inside the network.

Classical video transmission uses a fixed bitrate from the beginning to the end of the transmission. However, because network bandwidth varies from people to people, the video is better to be sent at the appropriate rate. Moreover, for the same user the bandwidth can vary during the streaming. For example, wireless networks use various network technologies with different characteristics, and they can change over time (interferences, mobility etc.) Another example is networks with shared bandwidth among several users, which could make the available bandwidth unstable. So we have on the one hand videos with different qualities available in the network, and on the other hand a variable network bandwidth to stream them. The current trend in research and in industry is video adaptation, where bitrate of sent video is changed during the streaming itself. For instance, the new DASH standard uses HTTP on top of TCP. HTTP is available everywhere, even on phone networks and on very restrictive computer networks, and currently the connectivity is very important.

Additionally, more and more network applications, for example real-time media like audio and video streaming, can accept a certain level of losses. If they use TCP (Transmission Control Protocol), they have to pay the price for full reliability, with great latency. On the other hand, UDP (User Datagram Protocol) [150] lacks congestion avoidance support, which can eventually lead to congestion collapse, a situation which fear researchers on transport protocols. RTP (Real-time Transport Protocol) [173], while being a widely-used protocol for multimedia streaming, is an application protocol; as such, it is put on top of a transport protocol, such as TCP or UDP, hence it does not cope with transport protocol problems.

Another promising protocol for these applications is DCCP (Datagram Congestion Control Protocol), recently standardised as RFC4340 [116]. It can be seen as TCP minus reliability and in-order delivery of packets, two key points in video streaming, or as UDP plus congestion control. For our purposes, two interesting points of DCCP are that it allows choosing the congestion control used during communication and that it uses acknowledgements. Among the currently three standardised congestion control protocols, TFRC (TCP-Friendly Rate Control) is the most adapted to video streaming [77]. Also, acknowledgement packets give useful information to the sender, such as the lost packets and ECN (Explicit Congestion Notification) marks.

Choosing a transport protocol with congestion control avoids the above problems. However, this also means that the video data sending rate is regulated by the network bandwidth, and not by the application anymore. As the bandwidth is variable, a classical solution is to continuously adapt video sending rate to network conditions, method called *adaptation*.

The adaptation can be done in several ways. Usually, the video bitrate is increased or decreased each time the available bandwidth increases or decreases. Such adaptation, known in the literature as "rate adaptive video control", can be done by controlling video parameters such as quantization parameter, number of frames per second (FPS) and image size [50].

This chapter presents my work on adaptive video transmission using video streaming with congestion control. It can be grouped in three items:

- Propose an initial adaptive streaming architecture using RTP mixer and DCCP, and simulated with NS2, presented in [123] and updated in [124].
- Propose a method to adapt video to network conditions, called VAAL, presented in [159], and a generic method to avoid oscillations during adaptation, presented in [160] and with additional experiments in [161]. The mixed method with four enhancements to VAAL is presented in [162].
- Propose a taxonomy of parameters used in video adaptation, presented in [59].

## 2.2 Video streaming simulation architecture

### 2.2.1 Introduction

As written previously, video uses an important part of the bandwidth in Internet. RTP (Real-time Transport Protocol) and RTCP (Real Time Control Protocol) [173] are standards for streaming multimedia contents. RTP transmits the data while RTCP controls the RTP stream. Between the RTP server and the RTP client, two intermediate systems can be placed: the mixer and the translator. The mixer receives RTP packets, possibly changes the data, combines the packets and then forwards new RTP packets. A mixer can modify the data, for example, it can change the quality of the sound. The translator forwards RTP packets leaving their synchronisation source identifier intact.

Multimedia content streaming over wireless networks is facing several challenges: mobility, shared, limited and variable bandwidth. When a client moves, he may change wireless access points (Base Transceiver Station or WLAN Access Point) and multimedia contents must be redirected as quickly as possible. As the bandwidth always fluctuates, multimedia contents have to be adapted to the available bandwidth at a given moment. Moreover, clients use a wide range of multimedia devices and a client connected to a phone network with a smart phone will not be able to visualise the same multimedia content as a client connected to Internet with a laptop. This heterogeneity implies that one must be able to stream a wide variety of multimedia contents adapted to each case.

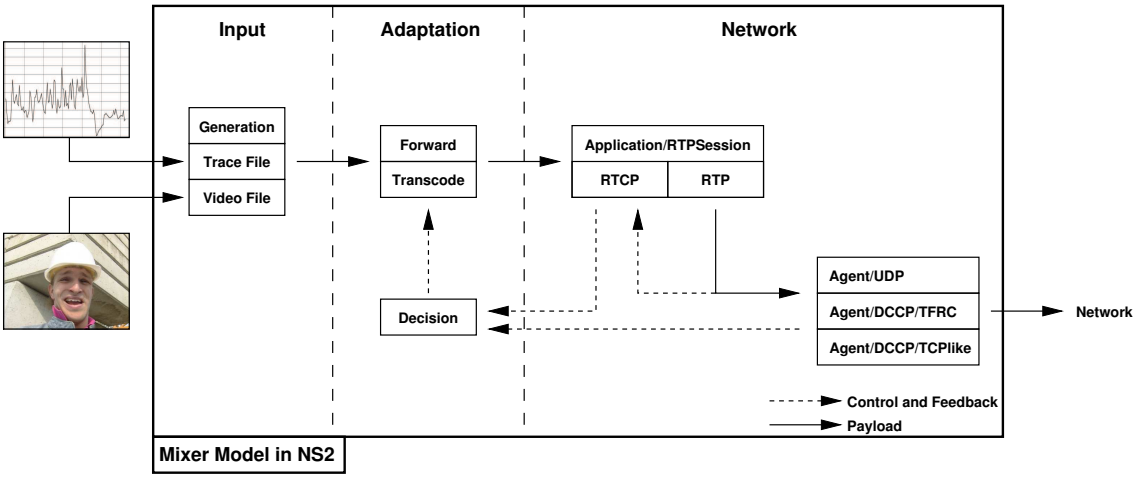



Figure 2.1: Mixer model in NS2.

Being able to simulate a video on demand (VoD) architecture, comprising a server, a mixer and a client, opens up real possibilities for the optimisation and comparison of elaborated strategies in a real context of flow concurrence. The congestion control used by the video must be TCP-friendly and allow moderate bandwidth changes, and we used TFRC [93] for that. The originality of our test bed is that it evaluates the final quality of video streamed into NS2. This is done by integrating new modules of real content streaming and PSNR (Peak Signal-to-Noise Ratio) calculation into NS2.

### 2.2.2 Related work

DCCP has successfully been used in video streaming. For example, [202] presents an implementation of TFRC in Linux, a codec and a video conferencing system with low latency, combining these elements. It divides the video system in three components: the codec, the DCCP module, and the algorithm deciding the video quality to use. The variables used for quality changing are: resolution, JPEG quality and frame rate.

[40] analyses losses in wireless links. The authors propose multiple connections for a video stream and deduce the following rule: "Keep increasing the number of connections until an additional connection results in an increase of end-to-end RTT or packet loss rate". Based on the RTT variation, the number of connections $n$ is increased by $\alpha/n$ or decreased by $\beta$, where $\alpha$ and $\beta$ are constant.

Several categories of video transcoding that modify the bitrate of the streamed video have been developed [120]. The first one is referred to as closed loop transcoding or Cascaded Pixel Domain Transcoder (CPDT) [201]. The video is completely decoded, possibly modified and then encoded, and this is the solution chosen for our mixer. The second category called Open Loop Transcoding (OLT) [68] do not completely decodes the stream but stops to the DCT phase. This solution saves CPU time but the resulting quality is not as good as the CPDT solution. A third category is an intermediate solution that pushes the decoding process deeper than OLT. It is named DCT Domain Transcoder (DDT) [208] and an implementation has been realised [166].

It is also possible to transcode by changing the resolution of the video [195, 21, 176] or by modifying the image frequency [200, 41].

### 2.2.3 Video streaming architecture

The general architecture of our simulation test bed is composed of two main parts: the mixer and the client.

Figure 2.1 presents the model of the mixer as it has been modelled into NS2. Three kinds of video data can be used as input of the mixer:

- Own generated data: The mixer generates its own data by using various algorithms. The

distribution of the various kinds of images (I, P and B) is given by Gismo [105].

- Real data: This mode allows to use real videos in NS2. The packetization is carried out by the mixer. The packets are sent through the different NS2 modules and the video is really transmitted between the server and the final client.

- Real packet sizes with dummy data: During a real transmission, the size of packets sent over the network is analysed and the characteristics of the packets are stored so that they may be used in NS2.

The input is sent to the core of the mixer, that is to say to the adaptation module, which decides either simply to forward the stream or to transcode it in order to fit the available bandwidth better. The packets are sent to the network module which comprises an RTPSession application which manages the RTP and RTCP agents, and the transport agent, for example, UDP or DCCP.

The client receives the packets from the network, and he can reconstruct the video exactly as if he had played it. This video is output to a compressed video file which can be compared with the original video. Finally, the PSNR can be calculated according to see exactly what the effect of losses or jitter is on the resulting video.

This simulation test bed allows to test different strategies to stream multimedia contents with various transport protocols. Besides, it also allows to see the effects of the network problems on the streamed contents clearly. Indeed, during multimedia streaming the lost packets will not have the same effect on the video quality. Some packets will seriously affect the visual quality of the video whereas others will not. This difference depends on various parameters like the type of lost packet, the time the packet is lost in the group of pictures (GOP) etc. In fact, if the last P-frame of a GOP is lost, the quality will not be affected because, just after this image, an intra image will be decoded. As the last P-frame of a GOP and the first I-frame of the following GOP do not have any time dependence, only one frame will be damaged. Another example is when a packet is lost on a P-frame just before a camera movement. The resulting image will be damaged but the camera movement will delete this error. That is why it is necessary to calculate the PSNR and not just to count the lost packets to evaluate the resulting video quality and this is the aim of our test bed.

**The mixer**

An RTP mixer [173] receives RTP packets from one or more sources, possibly changes the data format (for ex. on-the-fly transcoding), combines the packets in some manner and then forwards a new RTP packet. It is located the closest to the client in order to react as soon as possible to the variation of the bandwidth. As the clients are connected to a wireless network, the mixer should be ideally located in the bridge between the wired and the wireless network.

The various modules of the mixer are shown in figure 2.2:

- Client side: This module allows the mixer to be connected to the video servers.

- Server side: This module consists of the implementation of a complete RTP/RTSP server. The mixer is seen like a server from the clients' point of view.

- Buffer: The buffer is used to store a certain number of images before beginning the streaming to the client.

- Transcode/reassemble: This module is the element which makes it possible to adjust the quality of the stream according to the various constraints which act on the transmission. One of the major points of this module is the choice of changing the policy of adaptation.

- Decision: This module takes into account all the parameters which are given to it: information feedback from the server module (RTCP reports, for example) or information on the available bandwidth coming from the network module. Afterwards, it chooses the most appropriate video for the client according to the available bandwidth and the available video quality from the servers.

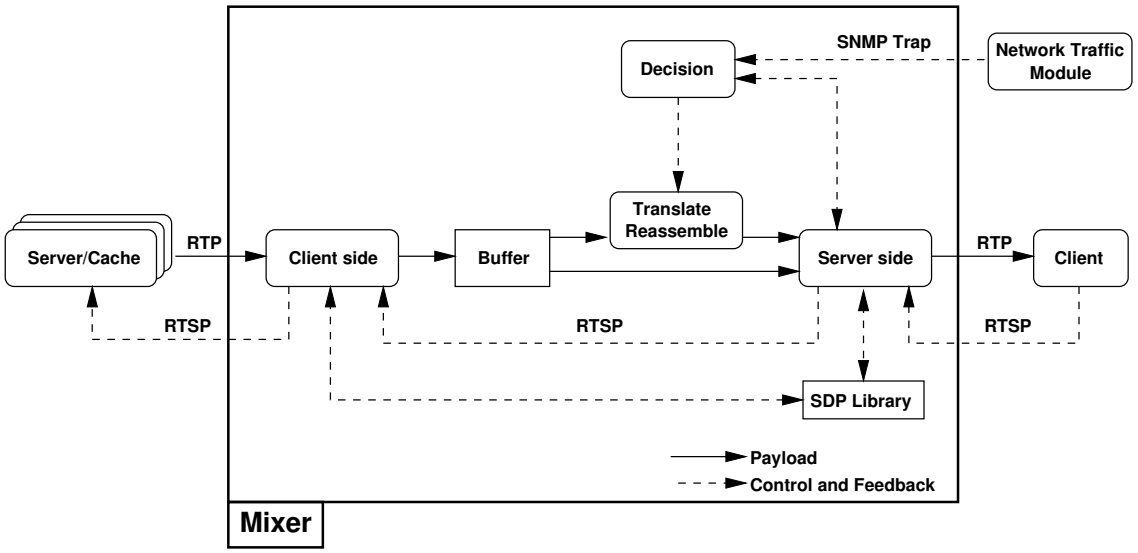



Figure 2.2: Internal architecture of the proposed mixer.

A typical session works as following:

1. A client connects to the mixer by the intermediary of the server module and requests the visualisation of the video.
2. The request is transmitted to the decision module which will ask the global architecture for the available quality for the required video.
3. The decision module will choose the video which is the most adapted to the client's characteristics and to the constraints of the network.
4. The client module requests the chosen video from the selected server. In the best case, a video the quality of which meets the needs, is directly available in the system and no transcoding or adaptation is necessary. If it is not the case, the decision module chooses the right transcoding method.
5. As soon as the stream is received by the server module, it is sent directly to the client.
6. If a quality change is necessary, the decision module asks the video server if a more adapted video is available. But in order to adapt the quality as soon as possible, it asks the transcoding module to change the quality of the stream while waiting for a new one.
7. As soon as the new video is available, the mixer stops the transcoding and streams the new video.

### 2.2.4 Simulation results with real data

The Network Simulator [140] version 2.28 is used for simulations. We use the shadowing model, where packets are always received for $d \leq s1$, always lost for $d \geq s2$ and received with a probability for $s1 < d < s2$. A wired streaming video server, an access point (AP), and a mobile streaming client are created for the simulation. Note that both AP and mobile use retransmission times. This simulation corresponds to a man walking on the street while watching News on Demand (NoD). The mobile moves according to the following scenario. Initially, it is at the AP. At t=0 s is uses a linear movement and goes far away the AP so that at t=100 s it is at 300 m. It stays motionless from t=100 s to t=150 s. At t=150 s it goes nearer the AP using a linear movement and same speed so that it arrives at AP at t=250 s. Between the stillness time, the mobile is located in the edge of the covered area of the AP, where most of the retransmissions appear.

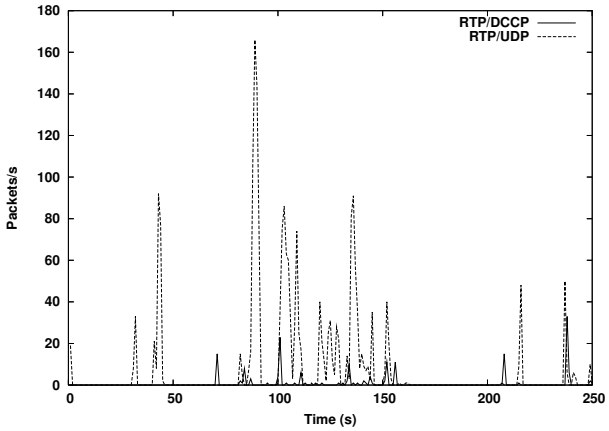



Figure 2.3: Packet loss results.

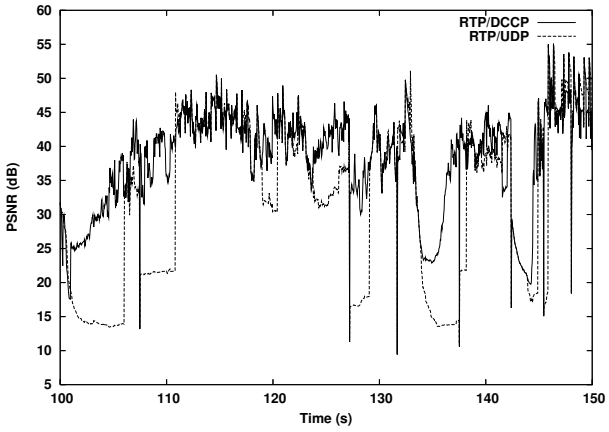



Figure 2.4: PSNR comparison results.

We transfer three times the same video with the same movement scenario. Only the transport protocol differs from one simulation to another. First, we use RTP over UDP, the original video streaming transport protocol. Then we use DCCP/TFRC with video adaptation and at last we propose to study: DCCP/TFRC with RTT modification.

In figure 2.3, during the $250s$ of RTP/UDP video transmission, 2063 packets are lost. With DCCP/TFRC, this number is reduced to 180 packets only. In the UDP case, most of these losses appear during the stillness time, when the available bandwidth is smaller than the video bitrate.

To compare the videos received by the client, we use the Peak Signal-to-Noise Ratio (PSNR), commonly used for video quality estimation. For each simulated transport protocol, the original video and the received one are compared in figure 2.4. Because of the TFRC and the video adaptation, the video read by the client on his mobile player has a better quality.

Detailed information about these results is given in [124].

### 2.2.5 Conclusions

This section proposed a complete video streaming architecture which includes a mixer combining a server and a client, and a mobile client. This model can help to optimise and to implement a dedicated congestion control protocol.

The measured improvements of our solution compared to the classical video streaming one are significant. Two types of results have been presented, for the network (throughput and packet loss) and for the visualisation of received video (PSNR). The PSNR results of our solution is really better than the classical solution RTP/UDP.

## 2.3 An oscillation-free method to adapt video to network conditions

### 2.3.1 Introduction

As already written, for video streaming an adaptation of the video to the network characteristics is very important. A cooperative approach between application layer and transport layer can improve the video quality perceptible by the final user.

The adaptation method proposed here (VAAL, Video Adaptation at Application Layer) uses transport protocol buffer overflow as a solution to find out the available bandwidth and to adapt the video content bitrate to the discovered bandwidth. Each $n$ fixed seconds, the server application computes the number of packets which failed to be written to the socket buffer. This number is used to control the video bitrate afterwards. A high number means smaller bandwidth and smaller bitrate. Zero error indicates either a stable or a greater bandwidth, so the bitrate of sent video could be increased. In this way, the bitrate of the watched video is fixed during each period of $n$ seconds, and can change only between periods.

Changing the bitrate as suggested above allows to optimise the *application* part of the video streaming. For yet better results, these methods could be coupled with other methods. For example, a well-known problem with losses in wireless networks is that they cannot be differentiated from congestion losses, hence the sender reduces the throughput while it should not [163]. Another optimisation on the *network part* useful on lossy links is the FEC (Forward Error Correction) ([182] for example). Also, *video-specific network techniques* allow for example to prioritise [90] or retransmit [99] only important packets (I packets in an MPEG-encoded video) on the server side. A commonly-used such technique is ALF (Application-Level Framing), which cuts intelligently video data in packets (avoiding for example to put one small video frame in two packets, which would be more sensible to packet loss); this is to be used in conjunction with the MTU (Maximum Transmission Unit) of the network.

It is important to understand that our method is only a small part in the optimisation chain for video streaming, which can be used and is beneficial independently of the other methods presented above. As such, *we consider in this work only network parameters and network performance criteria.*

The rate control is not a new idea to ameliorate video transmission. In the literature, there are a multitude of papers about this topic. Most of them were written 5–10 years ago, where solutions using new/modified transport protocols were used (such as [111]), which *are not deployable in reality* and risky from the congestion control point of view. Moreover, only few of them use experiments. The only papers we have found about rate control over DCCP are [94, 121], and they use simulations[1]. A new trend in adaptive video is also to use HTTP over TCP, such as the recent ISO MPEG DASH standard [100], which has advantages, but also drawbacks, such as, in videoconferencing, mandatory 100% reliability given by TCP use.

In this context, this is the first time that the *buffer overflow method is analysed for video adaptation.* Moreover, this is, to the best of our knowledge, *the first paper which uses DCCP in real experiments of video streaming in wireless networks.* Our solution is *very simple to deploy*, as only the application on the sender side needs to be modified (does not need to modify the receiver, nor the transport protocol). As a corollary, our method works with any transport protocol which has a congestion control.

The scope of our method is unidirectional and bidirectional communications, such as VoD (Video on Demand) and videoconferencing. Our method applies ideally to videoconferencing, because it is delay-sensitive and it consists mainly in a few simple additions/divisions per second and updating the value of the quantisation parameter; also, there is no need of support, such as caches, from CDN (Content Delivery Networks).

[1] There are serious concerns about simulation results, such as "around 50% of the papers appeared to be... bogus" and "who has ever validated NS2 code?" (from September 2005 archives of the e2e-interest mailing list). This is especially true for wireless link simulation.

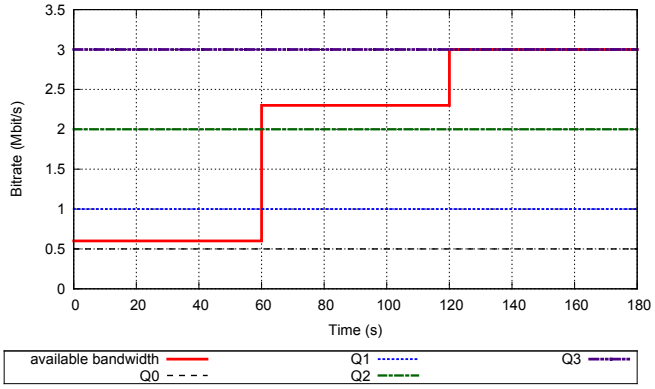



Figure 2.5: Example of available bandwidth vs available bitrates.

### 2.3.2 Advantages of video adaptation over static encoding

Video adaptation is useful when the available bandwidth varies during transmission. This appears in two cases: either the network bandwidth itself changes, or the bandwidth available for the flow changes.

The first case appears in wireless networks, where data rate changes according to radio link quality, and the bandwidth varies often. Data rate decreasing has multiple reasons:

1. interferences due to environment or to presence of another equipment working on the same range of frequencies, which decrease the signal to noise ratio for a short period;

2. mobility which, depending on the distance between the mobile and the access point, leads to signal attenuation and might also cause dynamic rate scaling.

This case also appears when an ISP (Internet Service Provider) changes dynamically the bandwidth allocated to a user. Indeed, with the increasing number of streaming services on Internet, more and more traffic is generated, which leads ISPs to voluntarily limit user bandwidth even during a transfer. We simulate this case by using a traffic shaping model as described in experiments section.

The second case is when a variable number of flows share the same path. The bandwidth available for video streaming is also variable for each flow. We illustrate this case by using a different number of concurrent flows as described in our experiments section.

For these cases, the adaptation helps to take advantage of all the available bandwidth or to avoid numerous lost packets.

#### Case study

Classical video transmission uses the same bitrate from the beginning to the end of the transmission. For comparison, we give some theoretical results based on a bandwidth between sender and receiver which changes according to figure 2.5. It simulates the limits imposed by ISPs, but also (on a larger scale) the dynamics of the available bandwidth between two hosts on a network. As already known, many factors contribute to this dynamics, for example when a user goes further or nearer the access point in a wireless network, or when more or less packets are injected into the network. The bandwidth changing pose problems to the video transmission because when bitrate is smaller than bandwidth, the network is underutilised, and when bitrate is higher than bandwidth, packets will be lost.

The figure presents the available bitrates of a hypothetical video: 512kb/s (Q0), 1Mb/s (Q1), 2Mb/s (Q2) and 3Mb/s (Q3). Also, as shown in the figure, during the 180 seconds of the video transmission the bandwidth changes three times: 600kb/s for the first minute, where 512kb/s video bitrate is the best choice; 2300kb/s for the second minute, where only quality of 3Mb/s should not be used; and 3Mb/s for the last minute, where any video quality could be used.

In this example it can be noticed that none of the four available bitrates is suited for the whole transmission:

| *Quality* | *Sent pkts* | *Received pkts* | *Lost pkts* |
|---|---|---|---|
| Q3 | 69120 | 45312 | 34.4% |
| Q2 | 46080 | 35328 | 23.3% |
| Q1 | 23040 | 19986 | 13.3% |
| Q0 | 11520 | 11520 | 0% |
| Ideal | 42240 | 42240 | 0% |

Table 2.1: Number of sent and received packets for each static bitrate and the ideal case.

1. Q0 is good for the first minute but choosing it will prevent the user from fully utilising the bandwidth for the remaining time.

2. Q1 is not good for the first minute and has the same drawback as 512kb/s bitrate for the remaining time.

3. Q2 is not good for the first minute; it could be chosen for the second minute, and has the same drawback as 512kb/s bitrate for the remaining time.

4. Finally, Q3 is not good for the first and second minutes; it could be chosen for last minute.

In order to compare the qualities, we compute for each of them the number of sent packets, the number of received packets and the percentage of lost packets. The results are given in table 2.1. For example, for a packet size s=1024 bytes, quality Q1 information is obtained like this:

- 1Mb/s / (s=1024*8b) * (t=180s) = 23040 sent packets;
- 0.6Mb/s / (s=1024*8b) * (t=60) + 1Mb/s / (s=1024*8b) * (t=120) = 19968 received packets (Q1 is greater than available bandwidth for the first 60 seconds, so only 600kb/s are received from the 1Mb/s sent);
- 23040 – 19968 = 3072, i.e. 13.3% lost packets.

The ideal case appears when the bitrate is adapted to available bandwidth, i.e. 512kb/s for the first minute, 2Mb/s for the second minute and 3Mb/s for the third minute. It leads to no lost packets and maxb*time sent and received packets, where maxb is at any moment the maximum bitrate less than or equal to the bandwidth.

First, this table confirms what we said before: no static bitrate is suitable for the whole transmission. Second, the adaptation is clearly better than each of the static bitrates. The only quality with no lost packets is Q0, but it is worse than ideal case, since it has only 11520 received packets compared to 42240 for the ideal case. Note that the number of received packets for Q3 is higher than the ideal case, since it leads to lost packets; e.g. for the first 60 seconds, the ideal case sends at 512kb/s and receives the same, while Q3 sends at 3Mb/s and receives 600kb/s.

Finally, it is difficult, or even impossible, for the user or some program to decide at the beginning of a video transmission which static bitrate is the best to use during the whole transmission. On the contrary, the adaptation is automatic, i.e. it does not involve any action from the user.

### 2.3.3 Related work

To avoid repetition, methods doing video adaptation are presented in related work section 2.4.4.

### 2.3.4 VAAL, video adaptation algorithm

To be able to know whether to increase, maintain or decrease the video quality, the sender application should have some information about the available bandwidth. In our proposed method VAAL, the bandwidth availability is measured through the number of packets which failed to be written to the transport protocol socket (buffer overflow). The higher the number, the less the available bandwidth. Regularly, VAAL computes this number of packets and changes the video bitrate accordingly. VAAL

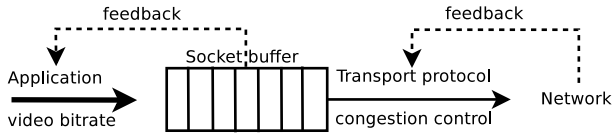



Figure 2.6: Video data flow on the sender.

requires a transport protocol with a network congestion control, no matter which one. It uses a simple algorithm. Finally, it is made at application layer and does not need any change to system kernel.

Note that while doing adaptation, the sender should try to minimise the quality fluctuation. It is not preferable, in our opinion, to change quality at each sent packet but on a longer scale (each 100ms, each 2sec or each I image for MPEG video for example). However, some codecs may have constraints on the time when the quality changes.

### VAAL explanation

As shown in figure 2.6, the server application writes packets to the transport protocol socket buffer at a rate equal to the current video bitrate. The transport protocol has a congestion control which gives the rate at which the packets leave the socket buffer (and afterwards the machine, and enter the network). VAAL goal is to adjust the video bitrate to the rate estimated by the transport protocol. Thus, VAAL algorithm is divided in two steps: *discovery of the network conditions* (available bandwidth based on buffer overflow) and *quality selection* (the action to be done). These two steps are executed for each period of $n$ fixed seconds.

The discovery of network conditions works as following. When the application generates packets at a higher rate than the transport protocol can send to lower layers (network), packets are buffered. If the buffer becomes full, the new packets generated by application will fail to be written (buffer overflow). Thus, VAAL monitors the available network bandwidth through the transport protocol socket buffer overflows when the application tries to write a packet in it. During each period, VAAL computes the percentage of these failed packets, which we will call *write failure percentage* (WFP). Consequently, WFP is an indication of the network conditions: the bigger the WFP, the less the available bandwidth.

The quality selection (adaptation) works as following. At the end of each period, VAAL reads the WFP (given by the first step) and acts like this:

- If WFP is null (no packets failed when written to buffer) VAAL chooses the next higher quality level (higher bitrate) unless the quality is already the highest.
- Elsewhere, if WFP is tolerable (smaller than 5%), the quality is maintained at the same level. ITU.T G.1070 [102] recommends that the end-to-end IP packet loss rate in video streaming should be less than 10%. Hence, we chose a threshold of 5% of packet loss rate at the sender buffer (WFP $< 5\%$), the other 5% being left to handle the network losses.
- Finally, when WFP is greater than 5% and unless the lowest quality is already in use, VAAL searches for the highest quality $q'$ which fulfils:

$$q' \leq q(1 - \text{WFP})r \tag{2.1}$$

  where $q$ is the current quality. In this formula, $q(1 - \text{WFP})$ represents the bandwidth available for the period which has just ended, while $r > 0$ is a parameter (*aggressiveness*) which allows us to choose a quality with a bitrate different than the available bandwidth.

  Normally, the next quality $q'$ should be smaller than $q$, but depending on the aggressiveness $r$, the quality could remain the same or even increase.

As mentioned before, VAAL requires a transport protocol with a network congestion control, no matter which one. Also, VAAL is especially useful in video conferencing (video on-the-fly) because there is no need to re-encode the video, just changing the encoding rate.

**Implementation**

We have implemented VAAL video adaptation at the application layer on a GNU/Linux machine with 2.6.35 kernel (without any change to system kernel). The period of time $n$ is 2 seconds. Packets which failed to be written in the socket buffer are deleted (i.e. they are not stored so that they can be retransmitted later), in order to reduce delay. The program uses DCCP as transport protocol together with TFRC as congestion control (DCCP was implemented in the kernel a few years ago).

Once VAAL was implemented, we also studied four enhancements. The first three show better performance while the last one gives similar results.

**Initial bitrate** Starting the connection with a high bitrate will cause transport protocol buffer to drop a high number of generated packets, which has a negative impact on video quality at the beginning of each video. "Slow-start" solutions for the bitrate at the beginning of a video streaming have already been proposed, but they are not generally applicable, because the adaptation can often be done only at coarse grain (e.g. each 2 sec.) So in VAAL we took a simple and general solution: we decided not to start the transfer with a high bitrate but with a small one. Our choice was also not to use the smallest bitrate because it will take a few seconds to reach the highest bitrate if the bandwidth is high enough to support it. In our current implementation of VAAL, the initial video bitrate used is 1Mb/s.

**WFP computation interval** WFP is the parameter used by VAAL to estimate the available bandwidth. We have considered two ideas for the interval of time on which WFP is calculated: use either the last RTT (Rount Trip Time), or the interval of time for the last $p$ sent packets; $p$ should not be smaller than 20, in order to allow a percentage of failed packets (WFP) less than or equal to 5% to appear, cf. the quality selection above. However, because the RTT may be very small (depending on network topology) and the number of packets during this period could be very small too, we choose instead a hybrid solution using both ideas, where the interval of time is max (last RTT, last 20 sent packets).

$r$ **aggressiveness value** $r$ represents the aggressiveness of the transfer. As previously shown (equation 2.1), the next quality chosen $q'$ is the highest available quality which verifies $q' \leq wr$, where $w$ is the bandwidth. If $r = 1$, the next quality will be at most $w$, while if $r = 1.1$ for example, the next quality could be as great as $1.1w$. However, in this latter case it does not necessarily mean that the quality will always be greater than $w$; if for example available qualities of the transmitted video are increasing as powers of 2 (512kb/s, 1Mb/s, 2Mb/s etc.), then the highest available quality smaller than or equal to $1.1w$ is somewhere between $1.1w/2$ and $1.1w$, with an average of $1.1w * 3/4 = 0.825w$, which is smaller than $w$. On the other hand, if all qualities are available (i.e. supported by a codec during an on-the-fly encoding), then the next quality will always be near or equal to $1.1w$, which is greater than $w$, hence not a good solution.

As an numerical example on the influence of $r$ to the chosen quality, let's suppose that the available bitrates are the ones above (512kb/s, 1Mb/s, 2Mb/s etc.), $r$ is 1.1, and that in some point of time the current bitrate is 1Mb/s and the measured WFP is 7%. According to equation 2.1 in previous section, $q' \leq 1\text{Mb/s} * (1 - 0.07) * 1.1$, that is $q' \leq 1.023\text{Mb/s}$. This means that the quality of the flow remains at 1Mb/s in that point of time. However, if WFP was 10%, then $q' \leq 0.99\text{Mb/s}$ and the quality would decrease to 512kb/s.

As a conclusion, $r$ should be chosen carefully and be based on the available qualities of the video: the higher the bitrate ratio between consecutive qualities, the higher the $r$ value. In the current version several values for $r$ have been tested and $r = 1.1$ is used.

**Real vs theoretic bitrate** It is known that sometimes encoders cannot encode a video to a specific bitrate, i.e. the bitrate of the video generated is not equal, but (more or less) near the bitrate asked. We denote by $q_t$ the theoretic (or asked) bitrate for a video and by $q_r$ the real (or generated) bitrate.

As previously specified, when VAAL is in quality selection step and after WFP calculation, next quality $q'$ (for the following 2 seconds) is chosen so that $q' \leq q(1 - \text{WFP})r$ (formula 2.1), where $q$ is the quality used in the past 2 seconds. $q$ and $q'$ can have two different meanings:

- For $q'$ (in the future): The bitrate for the next 2 seconds can sometimes be predicted (for example for video streaming using pre-stored files), sometimes not (for example for interactive video conference, when the codec is not very precise). If prediction is possible, $q'_t$ or $q'_r$ can be chosen, otherwise only $q'_t$ can be chosen.

- For $q$ (in the past): At any moment, the real bitrate $q_r$ used for past 2 seconds is available. However, the theoretical bitrate $q_t$ used 2 seconds ago exists only if in the previous step ($q'$, in the future) it was chosen.

Since $q'_r$ is not available for any kind of video transmission, we prefer to choose $q'_t$ for VAAL. There is now a choice between $q_t$ or $q_r$; in the current version both values have been tested and they gave similar results, and we decided to *use $q_t$ for next quality decision.*

### 2.3.5 ZAAL, generic zigzag avoidance algorithm

A video adaptation method such as VAAL presented previously can sometimes lead to a continuous switching between two qualities: one smaller than available bandwidth and the second greater. For example, if a user is connected to Internet via a 2.5Mb/s link and the video is available in 2 and 3Mb/s qualities, then an adaptive streaming algorithm will constantly switch between these two qualities. Obviously, the best solution would be to stay with 2Mb/s much more time before retrying 3Mb/s quality. Since the adaptation algorithm does not know when the available bandwidth could be larger than 3Mb/s, it should retry a superior bitrate from time to time. This constant change induces what we call ZQS *zigzag quality switching* problem (the zigzags are also called oscillations). We noticed this phenomenon in our first experiments (see figure 2.9(a) for a clear example, where during the first minute the video bitrate is continuously toggling between 0.5 and 1 Mb/s).

If this issue is already known (see [4] for instance), only few papers treat it, and they do not solve it completely. For example, in [136] the receiver uses a back-off timer for each layer of the video. If a level led to lost packets, the receiver goes back to previous level and the timer for the level with losses is multiplicatively increased. Authors of [71] present a way for smoothing sent video bitrate and reducing the frequency of video quality changes. The main idea is that the application does not switch to a higher video quality until the sender is certain that the video will continue playing even after a reduction of the congestion window. A metric to evaluate the jerkiness of a video, given by the number of quality changes, is given in [73]. It uses a formula to calculate an effective frame rate for each video. This formula is used to limit the number of quality changes for each defined window during the transmission.

In this section we introduce a simple, easy to implement and scalable solution, called ZAAL (**Z**igzag **A**voidance **Al**gorithm), to this problem (for detailed information the reader is referred to [161]). We present also the integration of ZAAL in the method of video adaptation at application layer on the server (VAAL). VAAL behaves as explained in section 2.3.4 except that when the quality is to be increased (in the quality selection phase), VAAL consults ZAAL for whether a superior bitrate is allowed or not. If ZAAL decides that the increase does not lead to zigzag, then VAAL increases the quality; otherwise, VAAL maintains the current level.

Note that ZAAL is not an adaptation method. It is used only to prevent an adaptation method from frequently switching the video quality. The only information ZAAL needs comes from the adaptation method, whether some bitrate causes lost packets or not.

#### Zigzag-avoiding algorithm overview

ZAAL algorithm works by avoiding constantly using bitrates higher than the available bandwidth. For that, it maintains an successfulness value for each bitrate, called *successfulness* in the following. When the adaptive algorithm considers to increase bitrate (and only in this case), ZAAL checks if

the successfulness of the higher bitrate is lower than a threshold, called $\beta$; if this is the case, a higher bitrate *cannot* be chosen. Otherwise said, application uses a higher bitrate $i$ only if its successfulness $S_i > \beta$. After this process, the successfulness is updated.

More precisely, ZAAL algorithm uses the *successfulness* value each time an adaptation period ends (e.g. each 2 sec. in case of VAAL). Successfulness value is calculated separately for each bitrate (with different weights), denoted by $S_i$, which indicates if bitrate of index $i$ can be used for the next period or not. As such, this value expresses application last attempts to use the corresponding bitrate. In brief, when a bitrate generated failed packets to the transport protocol buffer, corresponding successfulness value is greatly reduced; when a bitrate was successful, the successfulness value is greatly increased; finally, when a bitrate has been used for a long time, the successfulness value corresponding to the higher bitrate is slowly increased. As a general rule, the smaller the successfulness value, the more the corresponding bitrate caused failed packets and application must avoid using it.

The successfulness $S_i$ (where $i$ is bitrate index) of each bitrate, which changes according to the history of that bitrate, is calculated using an EWMA algorithm (Exponential Weighted Moving Average). Using EWMA allows to give greater weight to recent history compared to older history, since obviously current bandwidth is better expressed by recent bitrate usage than by older ones. Additionally, different weights are used, based on the bitrate involved.

At the beginning of a video transmission, all $S_i$ values are set to 1. Then they are calculated each time the application wants to adapt the video bitrate to the available bandwidth, using the following general formula:

$$S_i = (1 - \alpha/d)S_i + s(\alpha/d) \tag{2.2}$$

where:

- $s$ is the successfulness at the time of measurement (the current "observation") and can be either 0 or 1. 0 value is used when the bitrate did not give good results (hence its successfulness value will decrease). 1 value is used when the bitrate gave good results, either because the bitrate did not cause failed written packets, or because the bitrate was not used recently (hence its successfulness value will increase).

- $\alpha$ is the degree of weighting increase/decrease, a constant smoothing factor between 0 and 1. A higher $\alpha$ discounts older successfulness values faster.

- $d$ is a division factor allowing to speed up or slow down the value increasing depending on the bitrate involved. In our algorithm, $d$ has three values: 1, 2 and 4. For $s = 1$, the greater the $d$, the slower the increasing of the $S_i$ value. The value of $d$ depends whether the application increases, maintains or reduces bitrate.

$S$ values are arithmetic reals between 0 and 1. They can change in three cases:

1. First, when the application increases the video bitrate, i.e. the new bitrate $k$ is higher than the actual one $j$. This appears when the application does not sense lost packets for a while with actual video quality. In this case $S$ should increase ($s = 1$) for all bitrates $i$ lower than or equal to the current bitrate. Increasing speed must be high ($d = 1$). So:

$$S_{i|i \leq j} = (1 - \alpha)S_i + \alpha \tag{2.3}$$

2. Second, when the application maintains the bitrate. This happens either when some packet losses occur but their rate is acceptable to maintain the current quality $j$, or successfulness of the higher bitrate $k$ ($S_k$) is low and application avoids using it. $S$ value increases for all qualities but with different division factor values (different speeds). So:

   - $d = 1$ and $s = 1$ (big increase) for all bitrates $i$ lower than the current one $j$:

$$S_{i|i<j} = (1 - \alpha)S_i + \alpha \tag{2.4}$$

- $d = 2$ and $s = 1$ (small increase) for current bitrate $j$:

$$S_j = (1 - \alpha/2)S_j + \alpha/2 \tag{2.5}$$

- $d = 4$ and $s = 1$ (very small increase) for the bitrate $k = j + 1$ higher than the current one $j$:

$$S_{k(k=j+1)} = (1 - \alpha/4)S_k + \alpha/4 \tag{2.6}$$

3. Third, application decreases the video bitrate. This appears when the available bandwidth is not enough anymore for the actual bitrate (i.e. bandwidth decreases during the current period), or when the application tries a bitrate higher than the bandwidth. In this case, average value must be reduced for the current bitrate $j$, other bitrates average values do not change. Hence, $s = 0$ and $d = 1$ (big decrease). So:

$$S_j = (1 - \alpha)S_j \tag{2.7}$$

In our experiments we intuitively set $\alpha = 0.3$ and $\beta = 1 - \alpha = 0.7$. Other values were analysed but either they lead to high number of adaptation iterations for the algorithm to converge, or they minimise its effects.

**Discussion**

We present in this section some characteristics of zigzag-avoiding algorithm. To simplify results, we use $\alpha = 0.3$ and $\beta = 0.7$. We can notice the following:

1. At the beginning, average values are set to 1. They are greater than $\beta = 0.7$, so there is no restriction using any bitrate, i.e. application is authorised to use all bitrates.

2. Studying the decreasing phase of the algorithm we can notice:

   - As mentioned previously, when a bitrate is greater than the available bandwidth, application avoids to use it for a while by decreasing its successfulness average value. Based on equation 2.7, the first time this occurs, $S_i = 1 - 0.3 * 1 = 0.7$. Hence, a bitrate which causes failed writing packets cannot be used two consecutive times.
   - The smallest $S$ value for a bitrate $i$ can appear when it is used with $S_i = 0.7 + \epsilon$ and it leads to failed packets. So, its new $S$ value is equal to $S_i = 0.7 * (0.7 + \epsilon) = 0.49 + \epsilon$.
   - A higher bitrate does not necessarily mean a smaller $S$ value. When reducing an $S$ value for a bitrate $i$, higher bitrates do not change.

3. Studying the increasing phase of the algorithm we can notice:

   - The maximum number of times ZAAL prevents an application to increase bitrate occurs when the higher bitrate $k$ has the minimal $S$ value of $S_k = 0.49 + \epsilon$ (see previous point) and ends when $S_k$ becomes $> 0.7$. Using equation 2.6, this corresponds to 7 unsuccessful attempts by the application for the higher bitrate $k$, followed by an 8th successful attempt. Figure 2.7 shows the evolution of this value: there are 7 unsuccessful attempts between $0.49 + \epsilon$ and a value greater than 0.7.
   - $S$ values between 0.7 and 1 are only possible when the bitrate does not cause failed packets in two cases: it increases slowly beyond 0.7 when the quality is stable (see equation 2.5), or it increases quickly beyond 0.7 when higher qualities are possible (see equation 2.4).

4. Finally, using an exponential average rather than a linear average allows to remember past values for a longer time when increasing average values $S$, and to give more weight to recent history than to old history.

ZAAL is a simple and easy to implement algorithm, consisting of 3 `if` clauses with a loop over all bitrates inside each clause. It is also scalable, since it has a linear complexity in number of bitrates (a loop on all bitrates), and also in number of clients (ZAAL is executed independently for each of them).


Figure 2.7: Successfulness average increasing during the biggest period of time.

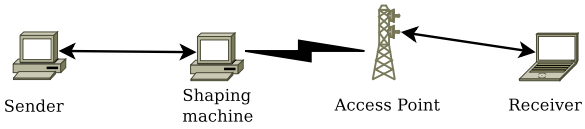



Figure 2.8: Network topology used for experiments.

### 2.3.6 Experimental results

#### Network topology

Figure 2.8 shows the real network used to realise experiments with the program presented in the previous section. The video streaming uses a real video of 180s, available in four qualities 3Mb/s, 2Mb/s, 1Mb/s and 512kb/s, as mentioned before.

Three series of tests are done. For the first series (traffic shaping series), just one flow is present at any moment during the video transmission, which can thus use all the available bandwidth. On the other hand, the output bandwidth of the shaping machine is changed three times to simulate a variable bandwidth: 600kb/s for the first minute, 2300kb/s for the second minute, and traffic shaping was stopped for the last minute (hence the output bandwidth is the original wireless bandwidth) (see figure 2.5). For the second series, two numbers of concurrent flows are present during the whole transmission: five and ten flows (called in the following *flows without gap* series). In fact, five flows at maximum bitrate are comparable to the bandwidth of the network (5*3Mb/s = 15Mb/s, which is about the bandwidth provided by a classical 54Mb/s wireless network), while 10 flows clearly exceed the bandwidth. In the third series, a various number up to twelve flows are present at the same time. Each flow starts 10 seconds after the beginning of the previous flow, except the first one which starts at time 0 (called in the following *up to twelve flows with gap* series).

During experiment execution, all the flows use the same algorithm (i.e. all of them use adaptation, or all of them use some fixed quality). The series above consist in executing the same experiments several times. First series has ten repetitions, and second and third series have five repetitions. Note that there is no retransmission for lost packets in all our tests (as given by DCCP).

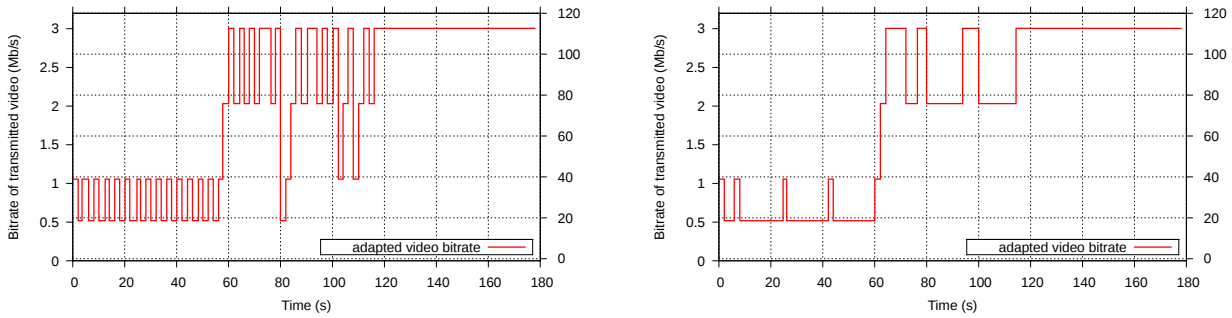
#### Results

Many experiments have been done [56]. This section presents only a summary of them.

First we confirm that ZAAL reduces indeed the number of oscillations. Figure 2.9 presents the results of one flow in case of traffic shaping. The x-axis represents the time from 0 to 180s, the duration of a video transmission, and the y-axis shows the video bitrate (VAAL with or without ZAAL).

We also investigate how ZAAL affects the transmission performance, namely the number of received packets and the number of lost packets. Table 2.2 presents numerical results of the experiments. It shows that using ZAAL, even if sometimes the number of received packets is lower, does indeed reduce the rate of lost packets while maximising the use of the bandwidth.

We pass now to VAAL+ZAAL (ZVAAL) method. First, we confirm that the bitrate adaptation takes well into account the DCCP buffer feedback (quality variation). As presented before, every two seconds ZVAAL looks at the write failure percentage (WFP) of DCCP buffer to decide if it has to




(a) without ZAAL: many zigzags occur

(b) with ZAAL: few zigzags occur

Figure 2.9: Quality adaptation for one flow in case of traffic shaping (using the *same* traffic shaping).

| | *Traffic shaping* | | | *10 concurrent flows* | | |
|---|---|---|---|---|---|---|
| *Method* | *Sent pkts* | *Rcv pkts* | *Lost pkts* | *Sent pkts* | *Rcv pkts* | *Lost pkts* |
| Without ZAAL | 47795 | 42043 | 5752 (12%) | 41191 | 32307 | 8884 (21%) |
| With ZAAL | 43548 | 39865 | 3683 ( 8%) | 36713 | 32477 | 4236 (11%) |

Table 2.2: Number of sent and received packets (average of all flows) with and without ZAAL.

increase, maintain or decrease its bitrate. We present here results for one flow in traffic shaping. For better visualisation, we show write *success* percentage, which is simply 1 – write failure percentage (i.e. 1 –WFP). The result of this test is shown in figure 2.10. As expected, during the first minute, where the bandwidth is limited to 600kb/s, ZVAAL keeps the bitrate at the lowest quality (512kb/s) while trying to sense a higher bitrate (1Mb/s) from time to time, e.g. at seconds 24 and 42. During the second minute, where the bandwidth is limited to 2.3Mb/s, ZVAAL discovers the new available bandwidth and uses most of the time a video bitrate between 2 and 3Mb/s (higher quality). In the last period the shaper is switched off, and ZVAAL sends the highest bitrate (highest quality). The last minute shows also that when bandwidth is higher than the highest quality, switching among qualities does not occur, hence ZVAAL is comparable to static transmission of the highest quality.

Further analysis of this figure confirms the useful properties of zigzag avoidance ZAAL algorithm. First, application waits for at least 2 periods of time before trying a bitrate which has recently caused losses (e.g. at the beginning, 1Mb/s did not work between 0s and 2s, hence it was retried not at 4s, but at 6s). Second, when a bitrate causes losses for many consecutive times, application waits more and more time to retry it (e.g. 1Mb/s at 0s, then at 6s, afterwards at 24s, and finally at 42s). Third, the maximum period during which the video quality was prevented to increase is 16s. (e.g. the first unsuccessful attempt finishes at 26s followed by the next successful attempt at 42s); during that time there were no losses (bandwidth much higher than bitrate), however ZAAL correctly prevented the bitrate increasing.

Next, in order to find out if adaptation is useful, we compare ZVAAL with the same application (using DCCP) but without adaptation (i.e. static encoding). We recall that our method optimises the network part of a video transmission, and can be used in combination with other video optimisation

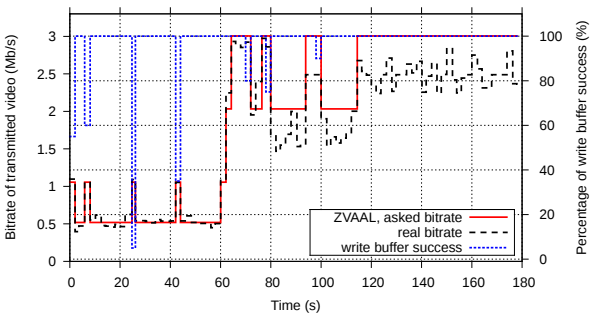



Figure 2.10: Quality adaptation for one flow in case of traffic shaping.

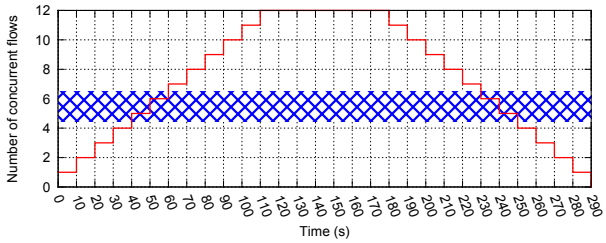



Figure 2.11: Number of concurrent flows at any moment for twelve flows with gap of 10 sec.

methods. As a consequence, the metrics used are the number of received packets and the number of lost packets. We assume that if a new method is able to maximise the number of received packets while minimising the number of lost packets, it will ameliorate the received video quality. The results for ZVAAL are taken from the tests done before. For DCCP without video adaptation, the same three series of experiments have been done, separately for each of the four qualities (Q3=3Mb/s, Q2=2Mb/s, Q1=1Mb/s and Q0=512kb/s). In the following graphics there are ten curves, which give the number of sent and received packets for each streaming quality: Q0, Q1, Q2, Q3 and ZVAAL (in order to distinguish them more easily, the curves for sent and received packets for each type use the same point style). Note that, even if all the curves are put on the same graph, the execution is done at *different* times. Also, even if the curves use lines for better visualisation, the flows and repetitions (on x-axis) are independent.

In this series of tests, up to 12 concurrent flows compete for the network bandwidth. This allows to compare the performance in a dynamic and more realistic situation where the number of flows varies over time. As a consequence, the best static bitrate cannot be known in advance, at the beginning of the transmission. This series of tests also mimics some characteristics of short lived requests/answers of the current Web, for example in these tests flows appear one after the other at the beginning, and disappear one after the other at the end of each test.

As already specified, there are about 10 seconds of gap between the beginning of each two consecutive flows. Figure 2.11 presents the starting time of the 12 competing flows and the number of concurrent flows at any time; for example flow number 7 starts at second 60, and there are 6 concurrent flows between seconds 50 and 60. Adding 180s to the start time gives the end transmission time for each flow, e.g. end time of flow 7 is 60+180=240s. There is also a blurred zone to indicate theoretically how many concurrent flows at highest quality (3Mb/s) the bandwidth can accept (4.5–6 flows). It can be noticed that the flows with the highest degree of concurrency are 6 and 7: during their lifespan the number of concurrent flows is between 6 and 12. The degree of concurrency experienced by a flow decreases as the flow number is further from 6 or 7 (flows 5 and 8 experience at least 5 concurrent flows, flows 4 and 9 at least 4 concurrent flows etc.) To conclude, as a simple rule, the less the distance between flow number and the middle (6.5), the higher the degree of concurrency.

Figure 2.12 presents one representative repetition. As expected, flows in the middle, e.g. flows 5 to 8, have the fewest received packets. Additionally, for ZVAAL the curve for the number of received *and* sent packets generally decreases from 1 to the middle (6.5) and increases afterwards, i.e. the flows generally send/receive packets based on the degree of concurrency they experience. This is contrary to static transmission (Q3 and Q2, without adaptation), where the flows in the middle experience a much higher dropped packets rate than others.

Overall average for all repetitions, shown in figure 2.13, confirms that ZVAAL gives similar results for all repetitions. The last column, Avg, is detailed in the table of the same figure. It shows the superiority of ZVAAL on all classical static transmissions:

- compared to DCCP Q3, ZVAAL has 7% fewer received packets ($40698/37683 - 1$), but 40% fewer sent packets ($1 - 41418/68623$), leading to 87% fewer packet losses ($1 - 3735/27925$);
- compared to DCCP Q2, ZVAAL receives 6% more packets ($1 - 35531/37683$) and sends 12% fewer packets ($1 - 41418/46818$), hence 33% fewer packet losses ($3735/11287$);
- compared to DCCP Q1 or Q0 it is much better, in terms of sent, received and lost packets.

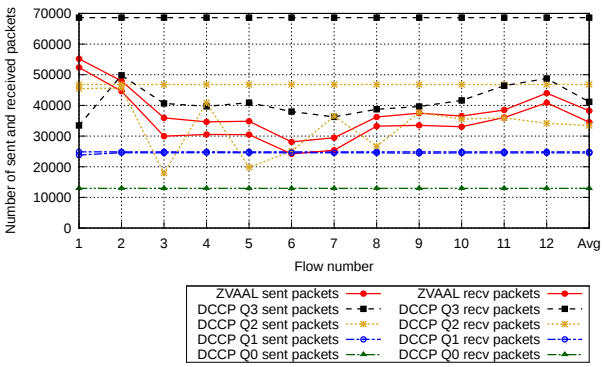
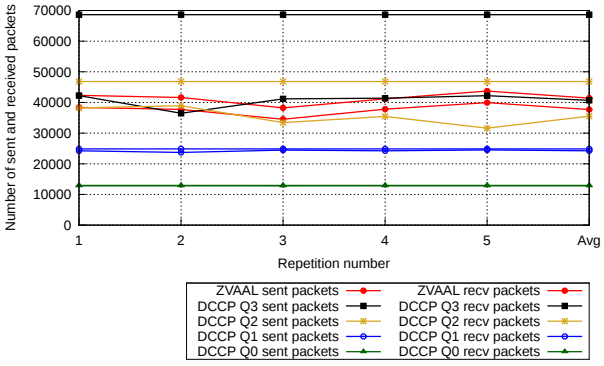



Figure 2.12: Comparison of number of sent and received packets for twelve flows with gap of 10 seconds (one representative repetition).




Average column in detail:

| | ZVAAL | Q3 | Q2 | Q1 | Q0 |
|---|---|---|---|---|---|
| Sent | 41418 | 68623 | 46818 | 24858 | 12938 |
| Received | 37683 | 40698 | 35531 | 24232 | 12828 |
| Lost | 3735 | 27925 | 11287 | 626 | 110 |

Figure 2.13: Comparison of number of sent and received packets for twelve concurrent flows with gap of 10 seconds (average of five repetitions).

As a conclusion, once again, with video adaptation (ZVAAL) the bandwidth is more efficiently used, especially when it changes dynamically or when the network experiences some bad conditions.

The final result is about ZVAAL fairness. We check if several applications on the same machine are fair with respect to the number of sent packets. A test with 10 flows is used as example. Figure 2.14 shows graphically the percentage of sent packets by each flow at application layer when using ZVAAL. More quantitatively, Jain's fairness index [103], defined as:

$$\left(\sum_i x_i\right)^2 / \left(n \sum_i x_i^2\right)$$

where $n$ is the number of flows and $x_i$ the number of sent packets of flow $i$, is equal to 0.9975, very close to the ideal value of 1. This result shows that ZVAAL maintains the fairness among concurrent flows on server (i.e. all flows have nearly equal percentage of sent packets). Note that the fairness presented here measures the fairness induced by our video adaptation algorithm, which we could call adaptation fairness. The absolute fairness depends on this adaptation fairness, and also on the fairness on the network level. The latter one is guaranteed in our method by using a TCP-friendly congestion control. In summary, given a fair (TCP-friendly) protocol in the network, the video adaptation fairness does not by itself unbalance this fairness.

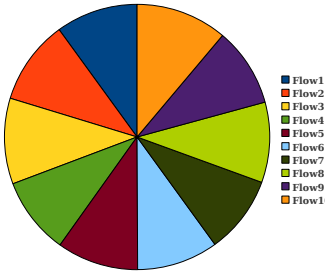



Figure 2.14: Percentage of sent packets by each flow at the application layer using ZVAAL: the percentages are nearly equal.

### 2.3.7 Conclusions

This section presented a simple but powerful method (VAAL) to adapt the content of video during streaming, using the buffer overflow method on the server at the application layer. ZAAL, a companion algorithm to reduce the number of oscillations in video quality, was presented too.

Intuitively, this method should lead to much better video streaming performance. Numerous real experiments confirm this hypothesis, i.e. the bitrate used during the adaptation is shaped by the available network bandwidth, and it generally either leads to much fewer packet losses, or avoids a under-utilisation of the network bandwidth. Moreover, the use of a transport protocol (DCCP in our implementation) with a congestion control (TFRC) guarantees the TCP-friendliness of our method, and TFRC makes it video streaming friendly.

## 2.4 Taxonomy of parameters used in video adaptation

### 2.4.1 Introduction

As already specified, video adaptation allows to greatly enhance user experience when watching videos. For that, many adaptation methods have been conceived. Some of them are sender-driven [119], others are receiver-driven [126]. The decision sometimes uses sender buffer [159], sometimes receiver buffer [119]. Some methods need beforehand data [73]; as a consequence, no method can deal optimally with all types of video transmission such as videosurveillance, videoconference, VoD (video on demand). There are a multitude of papers about this topic in the literature, however there is no article in the last ten years to classify the approaches used for adaptation.

In this context, this section fills this gap by classifying adaptation methods using a taxonomy of the parameters used for adaptation decision. This allows to better understand adaptive video and have a global vision on how it can be done. A valuable and realistic adaptation method is afterwards suggested, without further evaluation.

The scope of this section is IP-based networks. Among the many steps in video transmission over IP-based networks where optimisation can take place, such as ALF (Application Level Framing), FEC (Forward Error Correction), video encoding and compression, this analysis focuses on the adaptation step. We only take into account solutions which do video adaptation and present their specification in scientific or white papers. Also, we consider in this analysis *only network parameters and network performance criteria*. Moreover, we focus on adaptation caused by network conditions, so we do not take into account other parameters, such as CPU load, available codecs, resolution and other terminal capabilities discovery at the beginning of communication on the client side.

Note that this classification is done from the network viewpoint. As such, papers which deal with image information results (such as image encoding, image quality, subjective or objective evaluation of received videos) without taking into account network viewpoint are out of the scope of the current analysis and have not been taken into account.

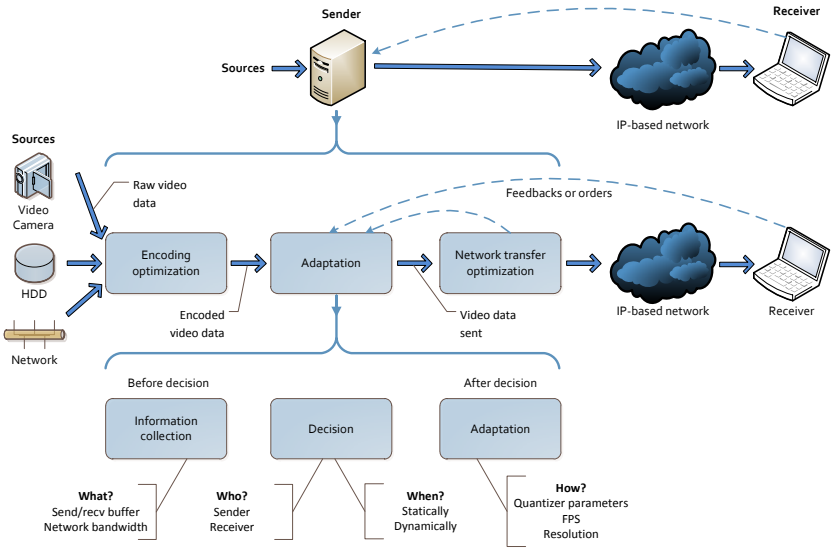



Figure 2.15: Classical adaptive video transmission, the parts where transfer optimisation occur, and the details of adaptation process.

### 2.4.2 Context

The goal of the section is to classify adaptation methods from some points of view. These points of view are detailed in the following.

Figure 2.15 presents a classical adaptive video transmission divided into three parts. The **upper part** of the image presents the classical figure of a closed-loop control between the sender and the receiver allowing to make video adaptation.

This is further detailed in the **middle part** of the image, which presents the video data transfer optimisations we have identified (we focus on end-to-end solutions, not on network ones, such as network resource reservation and service classes):

1. optimisation of encoding (video formats);
2. optimisation of adaptation;
3. optimisation of specific video transport protocols.

It should be noted that each of the three categories can have several optimisation methods. Also, a general video adaptation method can belong to several categories at the same time, i.e. it contains ideas which affect several parts of the transfer. Moreover, several methods can be used to further optimise video transmission.

The *first category* optimises specifically the video encoding in order to be the most adapted for the transfer over the network, and/or for other related purposes (such as storage on servers and encoder/decoder complexity). The signal processing community is constantly working to create new standards of video encoding, such as H.264 and its SVC extension, Theora and VP8, which increase the compression performance while maintaining a good video quality.

The methods in the *second category* act directly on the video data to be transmitted, for example by choosing the bitrate to encode the video. They can be divided in methods which add redundant information to video data, and methods which just change the bitrate of the video. For example, the former can use Forward Error Correction (FEC) [182] to protect the application against transmission errors at the expense of additional network resources used for this data.

The latter allows the flow to be network-friendly, and we will use the word “adaptation” for this. Adaptation can be done in three ways:

- Using stream switching techniques. The video is encoded in several streams (files) with different

qualities (bitrates) and the adaptation method dynamically switches among the streams so that the bitrate matches the available bandwidth of the network.

- Using layer switching techniques. The video is encoded into several, potentially orthogonal, layers: a base layer and enhancement layers. The adaptation method sends the base layer and as many enhancement layers as possible depending on available bandwidth.

- Using direct encoding techniques. The adaptation method re-encodes periodically the transmitted video based on the available bandwidth.

A widely-used layer switching technique is scalable video coding (SVC), which provides temporal, spatial and quality scalability. Using this technique, the video is stored as a single stream (file), which contains several enhancement substreams of the three scalability types. The particularity is that if enhancement substreams are removed, the remaining bit stream is still decodable, hence making the original stream scalable. For our purposes, removing substreams means not sending them on the network because of the adaptation process, otherwise said decreasing video bitrate. A drawback is that an SVC stream is less efficient in coding than an optimised single-layer stream at the same bitrate [174]. An SVC technique for H.264/AVC with low efficiency loss has been standardised [174, 192]. It leads to a high traffic variability [60], particularly suitable to adaptation.

Finally, the *third category* focuses on improving the network side of the transmission. Methods belonging to this category improve the way packets are sent without changing the video data itself. They act on many parameters. First, depending on the type of streaming, there are more appropriate transport protocols, such as DCCP (Datagram Congestion Control Protocol) [77]. Then, there are methods which selectively transmit packets (discarding some of them), based on available bandwidth [84, 90], or selectively retransmit packets, based on their importance (I frames in an MPEG-encoded video are always retransmitted, contrary to P and B frames, for example) [99]. There are also methods that improve the performance of transport protocols on certain types of network so that the rate of transmission is maintained at an appropriate level [163]. Furthermore, a commonly-used optimisation technique is ALF (Application-Level Framing), which cuts intelligently video data in packets (avoiding for example to put one small video frame in two packets, which would be more sensible to packet loss); this is to be used in conjunction with the MTU (Maximum Transmission Unit) of the network.

The focus of this analysis is to present the adaptation process which is shown in the **lower part** of the image of the figure.

We first note that the adaptation is *not* needed in some particular cases, for example:

- If the network bandwidth is relatively stable or at least is known in advance, then a sufficiently big receiver buffer correlated to a not so small startup time on the receiver removes the need of adaptation. The server however has to smooth data sending in order to match the available bandwidth, especially when the video uses VBR (variable bit rate) encoding. Several smoothing algorithms have been proposed in the literature, with various complexities and properties [197, 143]. They have been compared in [74], and the implications of smoothing on bandwidth and delay analysed in [61].

- Contrary to video communication (which is the scope of this study), real-time audio communication uses a small bandwidth. Thus, the authors of [165] make the implicit assumption that audio data will not exceed the available bandwidth and will not be lost. Hence, only the *delay variation*, given by RTT, needs to be addressed. They propose an ingenious mechanism to playout delayed packets, described in the following. The receiver uses a small buffer. It is assumed that audio data contains voice periods and silence periods. When voice data arrives in late, due to increased RTT, the receiver adds silence periods to a currently-played silence period. When RTT decreases back to normal, voice data arrives faster than its playout, and the receiver cuts from silence periods, thus restoring the normal delay. To resume, the client hears the same voice data, but with silence periods shortly longer or shorter.

  It should be noted that in video case there are no "silence" periods which can be shortened or lengthened. Additionally, the available bandwidth may not be sufficient, and considering the

RTT only is not good enough: if bandwidth becomes too small, packets get accumulated in sender or network buffers very fast and eventually are dropped there. Thus, in video case the problem is not only with the delay, but also with data packets which get lost.

In the general case where the adaptation process is used, it needs a decision about bitrate increasing, decreasing or maintaining. The adaptation process can thus be divided in three time frames: before, at and after adaptation decision. In the first time frame, all the necessary parameters needed by the decision are gathered. The decision is then taken at the second frame. Finally, once the decision has been made, in the third time frame the adaptation itself can take place. The three time frames are analysed below.

In the "before decision" time frame, several parameters exist to help taking the adaptation decision. For example, the available network bandwidth or the filling level of the receiver buffer. We group them in the *what* group, which is the main group in this time frame.

At the moment of decision, we take into account *who* does the decision (sender, receiver) and *when* it does it (e.g. each second). The decision is whether the bitrate should be increased, decreased or maintained.

Finally, after the decision, several parameters, grouped in the *how* group, can be changed. For example, modifying the bitrate can be done by modifying the quantisation parameter of the codec, or modifying the number of FPS (frames per second) or the resolution or other parameter, or changing the level in a multi-level encoded video. This can also take into account the client characteristics, for example the maximum resolution on that particular device or the set of video codecs supported.

Video adaptation is a multidisciplinary domain. The network research community pays more attention to the first time frame (information gathering about transfer speed) and the second time frame (adaptation decision), and the image research community is especially involved in the third time frame (video quality modification). The time frames can be optimised independently of each other. *This study focuses on the first two time frames, i.e. it does not present how video is changed to match the desired bitrate.*

### 2.4.3 Complexity of adaptive video transfer

Adaptive video transfer is more complicated to understand than it appears. One of the reasons is that different speeds are involved. As is shown in figure 2.16, several parts affect the speed of a communication:

- the sender: the application generates data at some speed (i.e. generating bitrate), and the transport layer may modify it by buffering and delaying it (i.e. sending speed);
- the network has its own speed, given by its resources and their instantaneous usage (i.e. data transmission speed);
- the receiver: the transport and/or application layer buffers data so that it is shown timely on the user screen (i.e. data consumption speed).

We analyse the variation of these speeds (i.e. whether they are constant or dynamic) in the following.

We assume that network congestion control is used. We consider that the amount of data can be measured either in bytes, or in video duration (seconds of video). During each second of the transmission:

- the sender application sends an amount of data either in terms of bytes or of seconds. It can send either a *constant* duration (one second of video, in a videoconferencing/videosurveillance case), or a *dynamic* duration (for example several seconds of data in a VoD case). On the other hand, it generally sends a *dynamic* amount of data in bytes (equal to current bitrate, which changes over time);
- similarly, the network transports a *dynamic* amount of data (either in bytes or in seconds);
- the receiver application consumes a *constant* video duration of data (one second of video), and a *dynamic* amount of bytes of data (equal to sender bitrate in an ideal network).

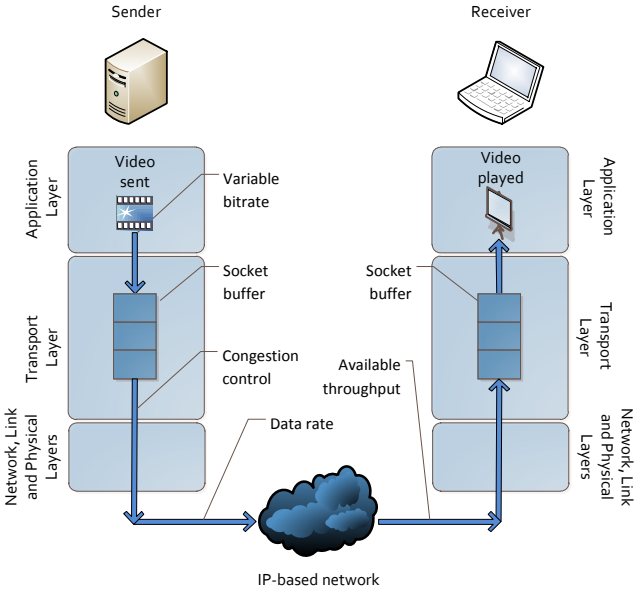



Figure 2.16: The various speeds involved in a video transfer.

| | Sender app | Network | Receiver app |
|---|---|---|---|
| Secs | const or dyn (ctrl) | dyn (non ctrl) | const |
| Bytes | dyn (ctrl) | dyn (non ctrl) | dyn (sender&net-ctrl) |

Table 2.3: The variation and controllability of the speeds involved in a video transfer.

On the other hand, the sender has control over its own dynamicity, since it can change the bitrate. The video system (sender and receiver) has no control over network speed since it is defined by available bandwidth, an external factor to the video system; the network is also responsible for lost packets; thus, network input (at sender) is not necessarily equal to network output (at receiver). The receiver consumes an amount in bytes at the speed given by the sender bitrate and network bandwidth, while the data duration consumed is constant (except if the user pauses the video stream). So the only uncontrollable factor comes from the network. This is summarised in table 2.3.

Given the discussion above, we end up with three different speeds (on sender, network and receiver) and two different measurements of speed (in duration or in bytes). All these parameters are correlated, for example when network speed increases, receiver data accumulates and sender buffers become empty, hence the sender speed can increase. As such, adaptation methods can use any of them, and that is the main reason why several types of methods exist. Next section presents some of them.

### 2.4.4 Some methods presentation

We have analysed numerous video adaptation methods in order to differentiate them. This subsection presents a few examples of video adaptation methods found in the literature with regard to the parameters used by adaptation. A longer presentation is found in [59].

**Methods using information from sender buffer** In QAC, Quality Adaptation Controller [52], the adaptation of the video is done on the server side, which stores the same video in several files with different bitrates. QAC uses HTTP and an application buffer. A worker thread continuously takes video packets from the application buffer and sends them to TCP. It monitors the size $q$ of the application buffer. It chooses the highest bitrate which makes $q$ greater than a predefined threshold $q_t$, while keeping at the same time the data rate under available bandwidth. So, the buffer has always data to be transmitted. The available bandwidth in QAC is modelled as a disturbance, as explained in [132].

**Methods using information from receiver buffer** The Buffer-driven adaptation method, described in [119], takes into account the occupancy of the receiver buffer to infer which video quality should be chosen and streamed. The occupancy $B_E$ of the receiver buffer is used at the sender by the following equation (presented in [118]):

$$B_E = B_C + \left(\frac{pktnum * s}{i} - ER\right) * RTT \tag{2.8}$$

where $B_C$ is the last buffer occupancy, $pktnum$ the number of packets, $s$ the packet size, $i$ RTCP (RTP Control Protocol) interval, and $ER$ the encoding rate.

**Methods using information from network** Kofler et al. [114] present a method that handles the streaming towards a wireless terminal. The adaptation method presented uses a platform for streaming based on MPEG-21 [35] to stream multi-layer videos based on SVC extension of H.264/AVC. It is applied on the server side and operates on two layers, application and transport layers, but other layers, such as the physical layer, can provide additional information. The video is divided in several units, using the generic bitstream syntax decription (gBSD) tool. The adaptation is done on a unit as a whole. This technique is similar to dividing the video in several independent parts. The adapted units will be later sent through the RTP protocol. For network bandwidth estimation the authors also implemented the equation used by TFRC flow while using a feedback system based on RTSP to deliver network information (input parameters of the equation of TFRC throughput) from the client to the server. This information is sent periodically (once per RTT, Round-Trip Time).

**Methods using information from sender/receiver buffer and network** In [142] the video is divided in several spatial layers, several temporal layers, and several quality layers with one base layer and several quality enhancement layers. The available bandwidth is estimated in two different manners. At the beginning of the transmission, the available bandwidth is estimated through a technique similar to PTR (Packet Transmission Rate) [97], which is a variation of the well-known *packet pair* technique. Afterwards, the client periodically sends to the server the estimated bandwidth $B$, the occupancy of the receiver buffer and acknowledgements for base layer packets.

**Methods proposed by major companies** Examples are IIS Smooth Streaming [204], Adobe HTTP Dynamic Streaming [64], HTTP Adaptive Live Streaming [145], and the recent standard MPEG DASH [100, 181]. All these methods are based on HTTP over TCP, and have some other common points. In all of them, the video is encoded in different bitrates and the resulting files are divided (either virtually upon client request or at content creation) to multiple independently decodable chunks, for example each interval of k seconds of the video (0-k, k-2k, 2k-3k, 3k-4k etc.) are stored in M files with M different bitrates. The general principle is that the client initially fetches a manifest file with information about available files (such as quality, duration and file name) and afterwards it regularly measures when data of the k seconds arrived to infer network capacity; for example, if it arrived in less than k seconds, then the network is faster than the current playback, hence for the next k seconds the client requests a file with higher bitrate. These requests use standard HTTP GET transactions.

### 2.4.5 Taxonomy criteria description

After reviewing several articles on video adaptation, we focus now on creating a taxonomy for adaptation methods. This section details the criteria used for the taxonomy. Some criteria deal with the context where the method can be used, an information especially useful to the video provider. Others deal with the parameters needed for the adaptation method (how the adaptation process works). Finally, each method presents some specific characteristics. Therefore, we divide criteria in three parts: context, functioning and properties.

Note that we take into account criteria related to the *adaptation idea*. We are not interested, for example, if the method was validated through simulations or real experiments, or if the examples given in the article were for videoconference or for VoD or for another video type. Moreover, we focus on before adaptation and at adaptation time frames, not on after adaptation.

**Usage context**

**Compatibility with ahead of time (beforehand) data availability:** whether the method works when no or few data is available on the sender before transmission. Based on the amount of data available (or data generation) before adaptation or sending process, we identify three cases:

1. no data is available: this is the case for videoconferencing and for critical videosurveillance (CCTV, closed-circuit television), where data must be sent immediately after its generation.

2. some data is available: this is the case for digital broadcasting television, where data is sometimes delayed by a few seconds (the so-called seven-second delay or broadcast delay), and for non-critical videosurveillance.

3. all the data is available: this is the case for VoD (Video on Demand) or files stored on server.

The next table presents an example of this criterion on three hypothetical methods:

| | Method 1 | Method 2 | Method 3 |
|---|---|---|---|
| No data | yes | no | no |
| Few data | yes | no | yes |
| All data | yes | yes | yes |

Note that a method which needs no data for example works also when few or all data is available.

**Usage:** We consider here whether the method is appropriate to real-time video transmission (such as videoconferencing) or to delay-tolerant transmission (such as video on demand). A "real-time" method is of course appropriate for delay-tolerant transmission too. Also, if a method needs some beforehand data, then it cannot be real-time.

**Functioning**

**Who:** who takes the decision of the adaptation (whether it should increase, decrease or maintain video quality). Two values are possible for this parameter: sender and receiver.

This should not be confounded with who does the adaptation, which is always the sender, since the video is found on sender (for our purposes, adaptation on an intermediate network equipment, i.e. proxies, are included in sender part). It should not be confounded with who makes calculus either; the calculus can be made by several parties (for ex. sender computes the bandwidth and the receiver computes the packet delay), but only one party takes the decision. We do not take into account who makes calculus, since as far as we have seen calculus are simple and fast (such as summing up several values) compared to when the decision is made.

**What:** what information (parameters) are used to take the decision. Two values have been found in the literature: buffer (at sender or at receiver) and network. Both can be measured either in bytes, or in seconds of video.

**When:** when the adaptation decision (or parameter evaluation) is performed (note that the decision could be to change one or several video parameters, or simply not to change anything).

**Computing interval of parameters:** what interval in time is taken into account for the adaptation compared to the adaptation interval. Suppose the adaptation decision is taken each N seconds. The decision could use the parameters from the whole interval, a smaller one (for ex. last RTT), or even the instant value of the parameter. For example, each 2 seconds the average data rate on the last RTT is used to choose the bitrate for the following interval of 2 seconds.

**Properties**

The functioning of a method can need specific properties. For example, some methods use the well-established HTTP protocol, others work with any transport protocol with congestion control. Some methods work with with multi-layered video only, while others work no matter the video encoding. Some methods support MPEG-21, which is a standard allowing, among others, the sender to find out the characteristics of the client (codecs available, maximum resolution allowed etc.)

### 2.4.6 Discussion

Table 2.4 presents a classification of the methods reviewed based on the criteria specified in previous section. We discuss the results of each column, and afterwards make some general remarks.

**Beforehand:** In the beforehand column it can be seen that all the possibilities are present, i.e. some methods need no beforehand data, others need a few only, whereas other need all the video stream before starting transmission. As an example, methods with "all" work only in case of stored files (VoD), in which case they potentially give better results.

It is worthwhile to note that major companies solutions need all video data beforehand. Indeed, the manuscript file, which gathers information about all available video streams and allows client to regularly request the appropriate stream, is downloaded by client at the beginning of communication.

**Usage:** Some interesting points can be remarked. First, both values (real-time and delay) are found abundantly in the literature. Also, as written before, methods working in real-time also work in delay-tolerant transmissions.

Furthermore, methods based on TCP (such as [52, 73]) or which need few or all beforehand data (such as [65, 126]) are not considered to be real-time. Other methods, such as [191], use client-aware intermediary nodes for performing adaptation, and as such they are useful only for delayed video transmission. Some others, such as [194], use virtual buffers, a kind of aggregated buffer for buffers on server and intermediate nodes, which reduce lost packets rate but add delay.

We also consider that since all the methods presented adapt video, the moment when they adapt is not so important, i.e. adapting each RTCP interval or each 2 seconds does not prevent them to be real-time. We do not consider multi-layer encoding to have an influence on real-timeliness of a method either.

A final interesting remark is that major companies solutions need beforehand data, and as such they are not appropriate for real-time communications.

**Who:** In the table it can be seen that generally the decision is taken by the sender. This is not surprising, since most of the information used for the decision is found on the sender: the sender does the adaptation, it has the video data and information about them (available qualities/bitrates, size, encoding, future characteristics etc.) However, the solutions proposed by major companies, based on HTTP, are receiver-driven. In this case, the sender sends information about videos to the receiver at the beginning of the communication (through a manifest file), and the receiver regularly sends adaptation information to the sender (through HTTP requests).

The difference between research community, where the decision is taken by server, and industry, where the decision is taken by receiver, is interesting and can be explained as following.

First, as already said, video is found or generated on server, hence all the information about video is found on server. Taking the decision on server avoids such information to be regularly exchanged between server and client, so this appears as a natural and optimised choice in research community.

Second, industry is bound HTTP/TCP, and there are several reasons for this. As industry *deploys* their solutions at *very large scale*, practical concerns/constraints are extremely important (such as has *to work, now* on Internet). Avoiding interconnection issues allows major companies to attain as much clients as possible. Nowadays HTTP, which is on top of TCP, is the works-everywhere application protocol (it is allowed in all networks: Internet, telecommunication networks).

| | Context | | Functioning | | | | Properties |
|---|---|---|---|---|---|---|---|
| | Beforehand data needed | Usage | Who | What | When | Interval | |
| VAAL [159] | no | real-time | snd | snd buf, bytes | each $n$ sec. (2 for ex.) | last RTT or 20 last pkts | any TP with CC |
| QAC [52] | no | delay | snd | snd buf, bytes | when buf exceeds thresholds | instant | HTTP |
| Buffer-driven [119] | no | real-time | snd | rcv buf | each RTCP interval | instant | RTP, RTCP |
| MVCBF [125] | few | delay | snd | rcv buf, secs | when feedback arrives | whole interval | virtual buff, ML, RTCP |
| AHDVS [51] | few | delay | rcv | rcv buf, bytes | each 1.2 sec. | N/A | HTTP |
| Kofler [114] | no | real-time | snd | net, bytes | when feedback arrives | RTT | TFRC, RTP, ML, MPEG-21 |
| Eberhard [65] | all | delay | snd | net, bytes | at RTSP feedback | N/A | RTP, RTSP, ML, MPEG-21 |
| Wien [191] | all | delay | snd | net, bytes | when feedback arrives | N/A | RTP, ML |
| VTP [14] | all | delay | snd | net, bytes | each $k$ pkts | many previous periods | new transport protocol |
| Gorkemli [87, 89] | no | real-time | snd | net, bytes | when buf exceeds thresholds | a fixed interval | DCCP, ML |
| MPEG-TFRCP [188] | no | real-time | snd | net, bytes | each 32*RTT | whole interval | direct encoding |
| Evalvid-RA [121] | no | real-time | snd | net, bytes | each GOP | instant | DCCP TFRC |
| CBVA [73] | all | delay | snd | net, secs | each GOP | instant | TCP |
| DRDOBS [194] | few | delay | snd | net, secs | when buf exceeds thresholds | instant | virtual buff |
| Schierl [172] | no | delay | snd | net, secs | each RTCP feedback | instant | RTP/RTCP, ML |
| RAAHS [126] | few | delay | rcv | net, secs | each segment received | whole interval | HTTP |
| Nguyen [142] | no | real-time | snd | net, bytes & rcv buf | each GOP | instant | ML |
| Major companies [204, 64, 145, 100] | all | delay | rcv | net, secs & rcv buf | each packet received | whole interval/instant | HTTP |

Table 2.4: Classification of major adaptation methods found in the literature, grouped by *what* parameter.

Furthermore, solutions based on HTTP and with client-side decisions scale well: they can use HTTP caches already in place (because data for a given address does not change) and current content delivery networks (CDN), they yield a stateless protocol (the server does not keep track of clients), work well when client pauses/resumes the video (since the video is not unnecessarily transmitted during pause), do not overwhelm clients which do not have enough CPU power to process all data received from network, they work in IP multicast or with many receivers of the same video source etc.

It should be noted that HTTP by itself does not prevent the server to take the decision, as shown by QAC [52], where the decision is taken by HTTP server. However, this means that all the advantages on scalability presented above disappear. Also, this means that the server needs to be modified in order to check current network conditions. In QAC for example, the HTTP server fills an application buffer and a worker thread drains it by sending packets the TCP. As a side note, this also adds a small delay to transmission.

**What:** The *what* parameter is the essential and the most complex parameter. Indeed, to be able to know whether to increase, maintain or decrease the video quality, the decider needs some information about the transmission speed. Feedback from receiver is required, usually through a transport protocol with congestion control or through an application protocol such as RTCP or HTTP. We have identified in the literature two sources to get this information: from sender or receiver buffer, and from network. Both can be viewed in terms of bytes, or of seconds of video. They are explained in the following.

The first source of information is thus sender or receiver buffer. When the network is too slow, packets get accumulated on sender buffer. When it is too fast, packets get accumulated on receiver buffer. Thus, the filling level of those two buffers of transport protocols (also called *sockets* in programming languages) expresses transmission speed. Note that the use of these two buffers is not a new idea, since they are very similar to congestion window and receiver window used in classical congestion control and respectively flow control of TCP transport protocol.

The buffer can be measured in two distinct ways: either in *bytes*, or in *seconds* of video. For example, the same size of data in buffer, 1Mb, corresponds to 1 second of 1Mb/s video, and to 4 seconds of 256kb/s video. When buffer is measured in bytes, methods can look at the filling level of the buffer, or only whether the buffer is completely filled or not.

Classically, the adaptation using buffer information is done as following. On the sender side, a filling buffer (either in bytes or in seconds of video) means a network speed slower than data generation, hence the application should generate less data, so it should decrease video quality; and it should increase quality for buffer lowering. Conversely, on the receiver side a filling buffer (either in bytes or in seconds of video) means too much data arrived, hence network is faster than data consumption, hence sender can send more data by increasing video quality; a lowering buffer should cause a quality decrease so that an empty buffer (leading eventually to video freezing) be avoided.

From an implementation point of view, buffer filling information is currently provided by some operating systems and only for TCP[2]. Otherwise, the sender can "simulate" it by using its own application buffer as following: The sender puts packets in an application buffer, while another thread continuously moves data from this buffer to transport protocol buffer; thus the application has access to the filling level of application buffer.

The second source of information is the network itself: its speed, its delay or other related parameter. For example, when the network speed is slower than bitrate, or when $N$ seconds of video are received in $M > N$ seconds, packets do not arrive in time to receiver; when the network is faster than bitrate, the quality can be increased.

Again, the network information is divided in *bytes*, for example the available bandwidth or throughput, and *duration*, for example the time taken by some data to be transferred on the network between sender and receiver.

Network information can be obtained from transport protocols with congestion control, such as TCP ([73]) or DCCP/TFRC ([114]), or otherwise from application protocols, such as the classical RTP/RTCP ([192]). In the former case, transport protocols estimate bandwidth (or other network-

[2]On GNU/Linux, a `tcp_info` structure with such information is returned by `getsockopt` function with `TCP_INFO` as parameter.

specific parameter, such as instantaneous TCP congestion window size) which can be read at application level[3]. In the latter case, regular control feedback is sent from receiver to sender with information such as number of lost packets.

To resume, the sender application should decrease bitrate when the sender buffer fills up or when the network bandwidth lowers or when the receiver buffer lowers.

It is interesting to see what happens when the user stops the video for some amount of time (5 minutes for example). Methods which use the receiver buffer will notice this, by seeing the buffer filling up, hence they can for example increase quality until the best one and keep it sending. The other methods act as if nothing happened, filling up the receiver buffer. Nevertheless, this is subject to transport protocol constraints on receiver buffer, such as flow control in TCP.

It should be noted that the change in bitrate could be subject to constraints other than network-based. For example, VAAL [162] includes an oscillation-avoiding algorithm which prevents bitrate increasing if it leads to quality oscillations, and DRDOBS [194] uses rate-distortion optimisation, where the bitrate is changed only if the quality increasing is worth the extra bits. We do not detail such parameters because, as written in the introduction section, our study focuses on network parameters.

It is still an open question which *what* parameter is qualitatively better. Some methods (Nguyen [142] and HTTP-based [204, 64, 145, 100]) even use two parameters: receiver buffer and network. However, these parameters, either in bytes or in seconds of video, are dependent each other, which means that their combined usage does not give much more information than one parameter alone. For example, the fact that 2 seconds of video data have been received in 1 second can be measured in two distinct manners: measuring how much data has been transmitted (from the network perspective), and how much data the buffer has received (from the buffer perspective). Also, it is known that video bitrate equals data size divided by video duration, and network throughput equals data size divided by transfer time. This means that if in 2 seconds the receiver buffer received 1Mb of new data (buffer measurement), then the network throughput was 512kb/s (network measurement).

**When:** The *when* parameter deals with the time when the adaptation occurs. We were first tempted to divide this criterion in static or dynamic, however this does not work, since these two terms are always related to something: a method which adapts each N received packets, for example, is static in terms of number of packets, and dynamic in time. So we decided to consider this criterion as static in the following terms: in time (e.g. in seconds), in number of RTTs, in number of packets, or other. Mathematically, this gives a function f(t) = k*t, f(RTT) = k*RTT, f(nbpkt) = k*nbpkt, or f(...) Note that some codecs may have constraints on the moment when the quality can change.

This parameter has a notable characteristic. Doing frequent adaptations increases indeed video quality received by the user, but has an adverse effect. Methods where the *when* parameter is small (e.g. up to a few seconds) might create undesirable oscillations in quality. Indeed, constantly adapting and changing video quality often leads to a negative perception of the video by the user. Methods to avoid such oscillations exist [160].

As shown in table 2.4, several values are used in the literature for this parameter. Finding out the most appropriate value for *when* is still an open question.

**Interval:** There is no consensus on the value of this parameter. Some methods in the literature use an average on either the last RTT, or the whole interval. Others use the instant value, a fixed interval etc. Some articles, noted by N/A, have not presented it.

There are also methods, such as HTTP-based, which use several intervals, e.g. instant when using receiver buffer and whole interval when using network in seconds.

We are aware of the works on video bandwidth forecasting, in which the server introduces in the flow information about the following video frames, such as frame size, over an interval of one GOP for example, information which could subsequently be used by routers in the path to optimise the transfer [91]. It should be noted that this anticipation interval is not necessarily the same as the computing interval, which could itself be different than the interval where adaptation occurs, as explained in the previous section.

[3]See the previous footnote, on TCP.

**Properties:** In the same table it can be seen that some methods use HTTP. This allows to bypass firewalls in various, currently very restrictive, networks (Internet, 3G etc.) and provide a universal access service. Others use various transport protocols with congestion control (TCP, DCCP) or the classical feedback-featured RTP/RTCP. Others work with any transport protocol (TP) with congestion control (CC).

Additionally, several methods use multi-layer encoded videos (ML), which are currently seen much interest and provide good performance. Also, MPEG-21 standard support is provided by some methods too.

Given all the discussion above, we can wonder about what the *ideal method* is. Of course, it should be real-time, without beforehand data needed, so that it can be used in all cases, but this could sometimes be incompatible with other goals. In the following we propose a solution which we do not argue to be ideal, but could be valuable to study.

The parameters where the best value is still open question are *What*, *When* and *Interval*.

*Who* is a very important parameter. Methods based on server decision work in all cases, real-time and delay-tolerant transmission, since the server has information about video as soon as it is generated. On the other hand, methods based on client decision scale well, taking into account Internet resources (e.g. HTTP caches). A mixture of the two could potentially give the advantages of both solutions. For example, server sends small-size data information about video together with video data, and client takes the decision.

*Properties* parameter is very important too. Using HTTP solves connectivity issues at application level on currently-restrictive Internet. At transport level, TCP is connectivity-friendly too, but leads to non real-time behaviour. DCCP is also connectivity-friendly, since it is connection-oriented and has congestion control. Restrictive firewalls could be easily updated to allow DCCP. Therefore, a potential solution would be HTTP on DCCP for video transfers.

So our basic idea is that each HTTP transfer use TCP, as usually, except for video files, which use DCCP instead. Video files can be recognised using the extension of the file asked (for example .avi). Choosing DCCP instead of TCP can be done either on server (videos are always sent with DCCP) or on client (in the HTTP request header). Note that in DCCP data transfer is unreliable, but session establishment and teardown are reliable. We have already used DCCP (without HTTP) to stream video [159].

Summing up, a valuable solution we propose is to use HTTP on top of DCCP. The real-time goal could be obtained by making server send to client small-size data information about video together with video data, and let client take the decision about video adaptation.

### 2.4.7 Other surveys

In order to maximise the chances to find out related works, we have made a search using specific search terms and took into account all the articles returned. For the sake of precision, we describe here the method used.

An exhaustive search on *google scholar* was carried on March 2013 with the following search term: *allintitle: (multimedia OR video OR content) AND (adaptation OR adaptive OR adaptative) AND (survey OR classification OR taxonomy OR review OR comparison)*, which looks for articles containing in their title a word from each of the three OR groups[4]. Among all the papers found, we disregarded the ones not being a survey or which were completely irrelevant to our paper (for ex. papers about video *content* classification or which do not treat adaptation) or which were not written in English. Four papers remained, presented in the following.

Arsan [7] presents a review of bandwidth estimation tools for wired and wireless networks with respect to video streaming. The review is about estimating available bandwidth, which is one of the parameters used when doing video adaptation. This parameter corresponds to the *What* column of our analysis, when its value is *network bytes*, meaning that network information is used for adaptation. As such, it focuses on one particular part of our analysis.

[4]In practice, google scholar accepts only one group of ORs, so we divided the search term in 3*3=9 searches; for example one of them was *allintitle: multimedia adaptation survey OR classification OR taxonomy OR review*

Vandalore [186] is a very general survey on adaptive multimedia methods. It treats video compression standards such as MPEG and wavelet encoding, rate shaping[5], FEC, video data smoothing[6], video/audio synchronisation, operating system support (CPU time allocation, thread priority). Instead, we specifically detail parameters used in video adaptation.

Adzic et al. [2] consider the general case of Web page content adaptation (text, image, video etc.) to devices with specific constraints (small screen and bandwidth). They consider a full presentation, with video, audio, text, thumbnail etc. and the adaptation consists in cutting one of them (e.g. a proxy which removes all images from a Web page), modifying images so that they do not surpass screen size, adapting font size etc. The papers included in their survey do not adapt videos, but Web pages, contrary to our analysis. Hence, the two articles treat adaptation on different levels.

Pu [155] presents a survey on QoS mechanism meant for a distributed multimedia server, not for the transmission itself. He concentrates on middleware level, and considers various programming technologies, such as Java, CORBA and green threads to increase performance of multimedia server.

Sadaf [167] presents methods to choose a data rate (6, 9, ..., 54 Mbps) in 802.11 wireless networks function of the error rate of each data rate. The aim is to improve the video transmission from network point of view. It is specific to IEEE 802.11 networks: it takes into account channel availability, RTS/CTS and user mobility. This paper adapts network parameters, not data. Also, the scope of our analysis is broader (any network) and focuses on adaptation parameters.

### 2.4.8 Conclusions

This section presented several methods found in the literature aiming to adapt video to network conditions during video streaming. A taxonomy of parameters taken into account by decision methods has been introduced. Methods presented have been discussed with respect to the taxonomy.

It turns out that many parameters can be used when taking the decision of video adaptation. In the research community, usually the sender does the adaptation. Major companies solutions currently use HTTP instead, and the adaptation is receiver-driven. Three parameters are commonly used for the decision: sender buffer, network or receiver buffer. Values can be measured either in bytes or in seconds of video.

After method analysis, a valuable and realistic method for video adaptation, taking into account advantages from several methods, has been suggested. It uses HTTP on top of DCCP with a small traffic from server to client, who takes adaptation decision.

## 2.5 Conclusions

During the last 35 years (since congestion control in TCP appeared [152]), data transmission in networks has always been a timely computer science field. New network technologies with changing characteristics (e.g. wireless and long fat networks), new applications (e.g. video data, HTTP), new hardware characteristics (such as bufferbloat[7]) have fuelled this interest. A few transport protocols have been standardised the last decade, such as SCTP [180] and DCCP. It turns out that community has reached a point where changes, even beneficial, are difficult or impossible to make. Almost ten years after standardisation, DCCP, "intended for applications such as streaming media" [115], is almost nonexistent on Internet. Also, the one-year old MPEG DASH standard uses the old but widely-deployed HTTP and TCP protocols.

A relatively recent trend in the community is video adaptation for video streaming. Several types of methods have been conceived, with fundamental differences such as who, the server or the client or both, takes the adaptation decision and based on which information, still no cleaner winner has emerged from scientific or technical point of view. However, a standard recently has appeared, MPEG

[5]Rate shaping: adjust the rate of traffic generated based on the available bandwidth.

[6]Smoothing: in the context of VBR videos where different frames have different sizes in bytes, smoothing means sending parts of large frames later (or sooner), in the same time as smaller video frames, in order to use constant bandwidth.

[7]See for example [85] and `http://www.bufferbloat.net`.

DASH, which is embraced by major companies. After several years, the congestion control (used by DASH) has also won the battle with the absence of congestion control (solutions based on UDP).

I expect this field to become even more important, driven by the flourish of video images in our lives and the increasing interconnection of objects, with projects like google glasses, 3D video and ubiquitous videoconferencing, to name just a few.

# Chapter 3

# Communication in distributed intelligent MEMS

## 3.1 Introduction

Around year 2005, my computer science laboratory (LIFC, *Laboratoire d'Informatique de l'Université de Franche-Comté*) started to consider its integration to the FEMTO-ST institute. The institute was created in 2004 by the fusion of 5 laboratories in our geographical region (Energy, Applied mechanics, Micro nano science & systems, Optics, and Time-frequency) and later Automatics and micro-mechatronics systems laboratory too. As such, multi-disciplinary projects involving our laboratory and FEMTO-ST institute have been considered.

A member of my team in the computer science laboratory (Julien Bourgeois) created such a project, called *Smart surface*, which received funding from the recently created national research agency ANR, *Agence Nationale de la Recherche*, PSIRob (ROBO) program. More than twenty researchers from six laboratories were involved: our laboratory, FEMTO-ST institute, LAAS (Toulouse, France), LIMMS (Tokyo, Japan) and InESS (Strasbourg, France). The project received 465 keuros from 2006 to 2010.

I decided to get involved in this project and in the cooperation with the various partners.

The objective of the project is to design a distributed and integrated micro-manipulator based on an array of micro-modules in order to realise an *automated positioning and conveying surface*, as shown in figure 3.1. Each micro-module will be composed of a micro-sensor, a micro-actuator, a communication unit and a programmable processing unit. The cooperation of these micro-modules thanks to an integrated network will allow to recognise the parts and to control micro-actuators in order to move and position accurately the parts on the smart surface. The parts are small, they cover a few numbers of micro-modules (e.g. $4 \times 4$). Due to scalability, robustness to failures and even simplicity, we have conceived the communication network as a peer-to-peer network. The network consists of links between each two direct neighbours (4-connectivity). The communication must be faster than the sampling period. On the other hand all the measures must be done at the same time at each sampling period.

The smart surface was still in design phase when we worked on this project, and it was noticeably easier to construct a smart surface prototype at a greater scale than the micro-scale surface. A prototype of pneumatic surface with conventional technologies has been built (see figure 3.2), which, associated to a software emulating the distributed architecture, helped to conduct the proof-of-concept of future smart surfaces. The distributed air-jet manipulation surface is a $120 \times 120\ mm^2$ square surface with a 15x15 actuators array, upon which an object is moving in constant aerodynamic levitation thanks to the airflow that comes through the lateral common air inlet. Objects are moved by generating strong vertical air-jets through the other holes of the surface. A camera positioned above the surface provides the top view of the entire surface. It emulates the cell sensors and allows free discretisation of the part on the surface for testing purposes. The image processing calculates the state $b_i(k)$ of every emulated sensor $i$ for each video frame $k$ based on the grey level of pixels. $b_i(k)$ is 1 if the object is above the sensor $i$, and 0 elsewhere. The building of the surface and the control block have been done by another team, the AS2M department of our laboratory, hence they will not be presented here.

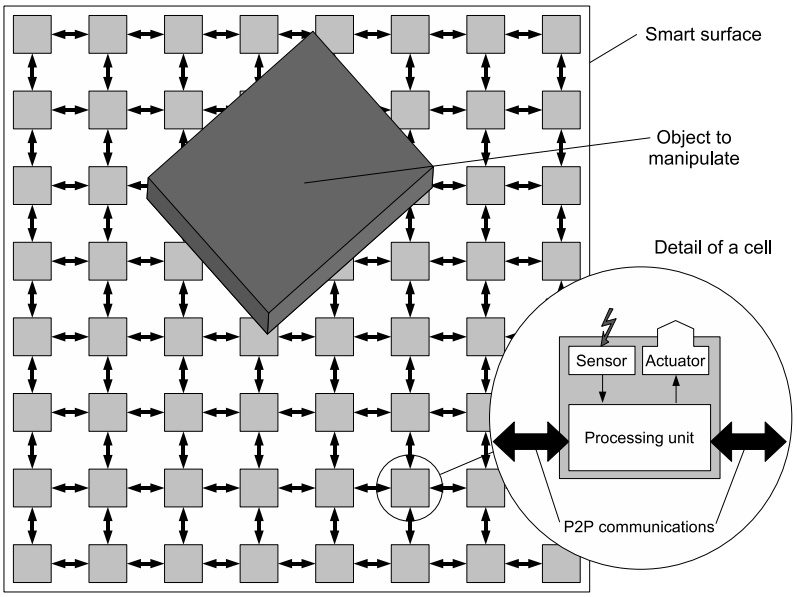



Figure 3.1: The smart surface concept.

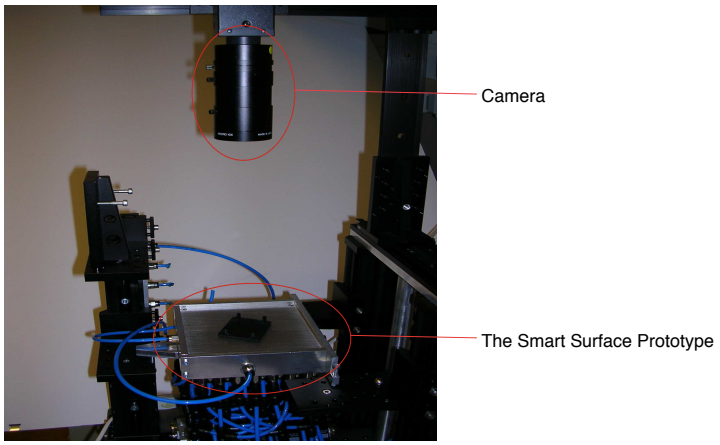



Figure 3.2: Overview of the smart surface prototype.

Inside this project, my computer science team is responsible for the information management inside the smart surface, i.e. distributed differentiation of the part and communication infrastructure. When on object is put on the surface, it has to be differentiated and afterwards the control plane must be informed about the type of the object, where it lies and where to move it; the control takes the "ball" afterwards. Some challenges of our team are pattern recognition for objects/parts of very low resolution (e.g. $3 \times 3$), real-timeliness required by control part of the surface, integration with other teams, and make it work with the *real* surface.

Working with people from other scientific disciplines is harder because their field is unknown to a computer scientist. In our project, some people usually worked with physical laws, and material fabrication, whereas we worked with computer programs or algorithms. The vocabulary also could be different. For example, a computer scientist would say that a *distributed control* means that the control is done by several components which communicate through messages, whereas an automatics scientist would say that it means that the system is discretised and have many points or variables.

This chapter presents my work on this project: a framework to test possible combinations of parts [31], a surface calibrator [32, 55], glued together with motion control [33], with enhanced part differentiation [66, 67] and information distribution using trees [171].

By the way, there have been numerous projects of MEMS actuator arrays in the past and more precisely in the 1990's. These pioneer researches have developed different types of MEMS actuator arrays, based on actuators either pneumatic [149, 82], servoed roller wheels [128, 129], magnetic [127] or thermobimoph and electrostatic [183]. Some of these preliminary studies use a sensorless manipulation scheme based on Goldberg's algorithm [86] for parallel jaw grippers. The jaw grippers are obtained with MEMS actuator arrays by creating opposite field forces which then can orient and move the parts. Böhringer et al. [23] have proposed a concept called "programmable force field" which is an extension of Goldberg's algorithm. This manipulation scheme which is well-adapted for jaw grippers has shown some limitations when adapted to MEMS actuator arrays. For instance, the absence of a command law can lead to uncertain behaviours [138] or MEMS actuator arrays have to be programmed for each different kind of parts. More recent research has been conducted in order to include sensors and to add intelligence to MEMS actuator arrays but it either fails to develop it at a micro-scale [20] or to be fully integrated [83].

More specifically, the literature on sorting and positioning micro objects in a low resolution context is, to the best of our knowledge, almost nonexistent. Low resolution has however been studied in other contexts, such as low resolution character recognition [101], and a novel approach based on the Radon transform for complex shapes identification [184].

## 3.2 Exhaustive comparison framework

As written before, the first step is to implement part differentiation algorithms on the Smart Surface. Before that, we are interested to find out criteria allowing high differentiation rates.

### 3.2.1 Framework overview

This section presents a framework for criteria comparison in differentiating parts, based on an exhaustive part generation. The framework is presented in figure 3.3. It receives as input a set of criteria, the maximum part size (a square) and the *number* of parts to differentiate. The framework exhaustively generates all the appropriate parts. It generates several comparison trees: differentiation tree, cost tree. An example of question which the framework answers to is: What criteria three random parts not greater than $3 \times 3$ do differentiate best?

The constraints of the framework, which will be relaxed in future sections, are: parts can be rotated only at 90°; we work on *family of parts*. We define a family of parts all the *ideal* parts which have the same image (discrete representation) on the surface. For example, the typographic letters L and L (with and without serifs) have the same image on the surface, because the serifs are much smaller than the sensors. Additionally, we consider there is no error in sensors and communications.

The framework has two phases: part generation and part differentiation.

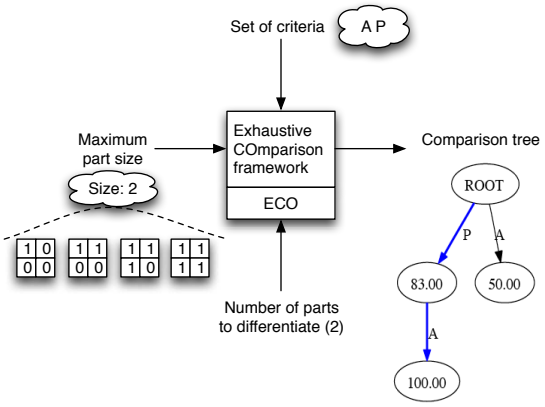



Figure 3.3: Overview of the framework.

| Max part size | Number of parts generated | Number of unique parts | Number of groups |
|---|---|---|---|
| $3 \times 3$ | 512 | 35 | $C^3_{35} = 6545$ |
| $4 \times 4$ | 65536 | 1280 | $C^3_{1280} = 348706560$ |

Table 3.1: Information for groups of three parts.

**Part generation** The parts on the smart surface are supposed to be represented by square matrices of size 3 or 4. In order to find criteria reaching 100% differentiation, all possible parts of size $P \times P$ with $P = 3$ and $P = 4$ are considered. This set of parts is used to generate groups of 3 parts. These groups are used to test the criteria or combinations of criteria which reach total differentiation. Part generation has four steps:

1. All the parts of size $P \times P$ are generated.
2. The resulting set of parts is reduced by eliminating translations, 90° rotations and mirrors (see table 3.1). Let $T$ be the total number of unique parts.
3. All the combinations of $n$ parts from the previously generated parts are generated (see table 3.1).
4. All the combinations of $CC_i$ criteria are generated. For example, if $C_i = \{A, S, P\}$ is the set of criteria, the generated combinations are $CC_i = \{\{A\}, \{S\}, \{P\}, \{AS\}, \{AP\}, \{SP\}, \{ASP\}\}$. This means that all the criteria are combined in order to differentiate the parts. Several criteria have been tested, presented in the next section.

**Part differentiation** For each criterion $C_j$ and each group $G_i$ of parts, a differentiation matrix $D$ is generated, with $D(i, j) = 1$ if the values of the criterion between the two parts $i$ and $j$ are different, otherwise it is 0.

$$\begin{aligned} &D_{G_i,C_j}(k,l) = 1, \quad \forall k, l \in P \\ &\Leftrightarrow C_j \text{ differentiates all } P \in G_i \end{aligned} \tag{3.1}$$

In the case of a combination of several criteria $CC_j$ the union of the differentiation matrices is computed. If the matrix $D$ contains only 1 values, the parts are said to be differentiated according to this combination of criteria.

$$\begin{aligned} &D_{G_i,CC_j} = \cup_{k \in CC_j} D_{G_i,C_k} \\ &CC_j \subset \{C_1, C_2, ..., C_m\} \end{aligned} \tag{3.2}$$

The matrix $D$ is upper triangular. A differentiation is said to be *total* if the matrices are differentiated according to *all* possible groups. Figure 3.4 is an example of the computation of a differentiation matrix with $G_1 = \{P_1, P_2, P_3\}$ for the combination of criteria $CC_1 = \{ASP\}$.

The major challenge is to find out a combination of criteria which leads to a total differentiation, i.e. for any group of parts, a differentiation of the parts using one combination of our criteria is always achieved.

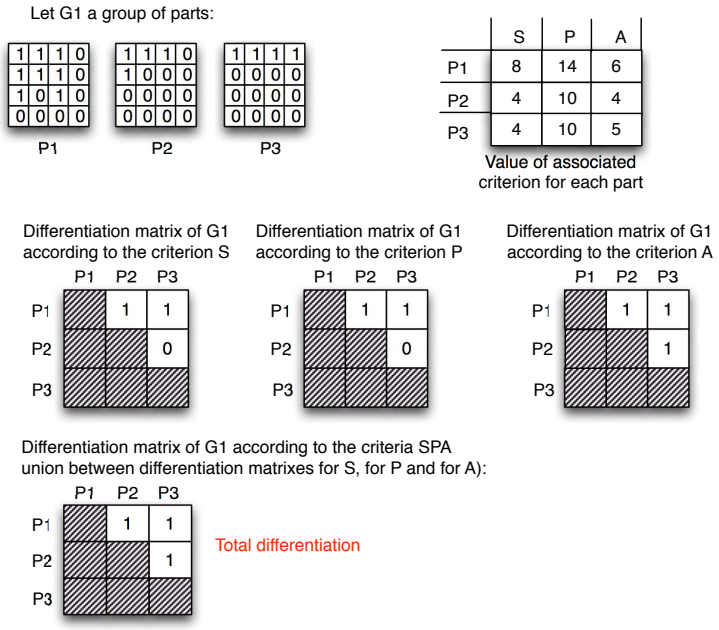



Figure 3.4: An example of a group differentiation according to a combination of criteria.

The following algorithm details this process. Let $n$ be the number of parts which need to be differentiated. The framework generates all the groups of $n$ parts. If $T$ is the total number of unique parts, there will be $C_T^n$ groups.

**for** each $CC_i$ = subset of $\{C_1, C_2, ..., C_m\}$ **do**
  **for** each group $G_i$ subset of $n$ elements in $P$ **do**
    **if** $CC_i$ is a criterion **then**
      build the differentiation matrix $D_{G_i,CC_i}$
    **else**
      $\{CC_i$ is a combination of criteria$\}$
      **for** each $C_j$ in $CC_i$ **do**
        build the differentiation matrix $D_{G_i,CC_j} = \bigcup D_{G_i,C_j}$
      **end for**
    **end if**
    compute the differentiation rate $t$
  **end for**
  compute the average $ta$ of all differentiation rates $t$
**end for**
build comparison tree

The usefulness of criteria is presented as a simplified tree (i.e. path $XY$ is the same as $YX$) called *comparison tree* (an example is in figure 3.5 later). Each node has a value expressed as percentage of differentiation using all the criteria of the path from the root of the tree. Finally, a cost (execution time, memory used etc.) is associated to each branch.

### 3.2.2 Numerical results

The aim of our work is to differentiate relatively small parts by finding a set of criteria. These parts are represented by square matrices of order 3 or 4. All criteria are tested in order to find combinations of criteria reaching total differentiation. Among these criteria, the fastest execution time and/or the lowest memory cost are selected.

The differentiation criteria must be simple and easy to implement. For example, the first criterion, $P$ (the perimeter), is the number of cell frontiers between "1" (pressed sensor) and "0" (unpressed sensor). The second criterion, $S$ (the area), consists in counting all the "1" contained in a part. Here

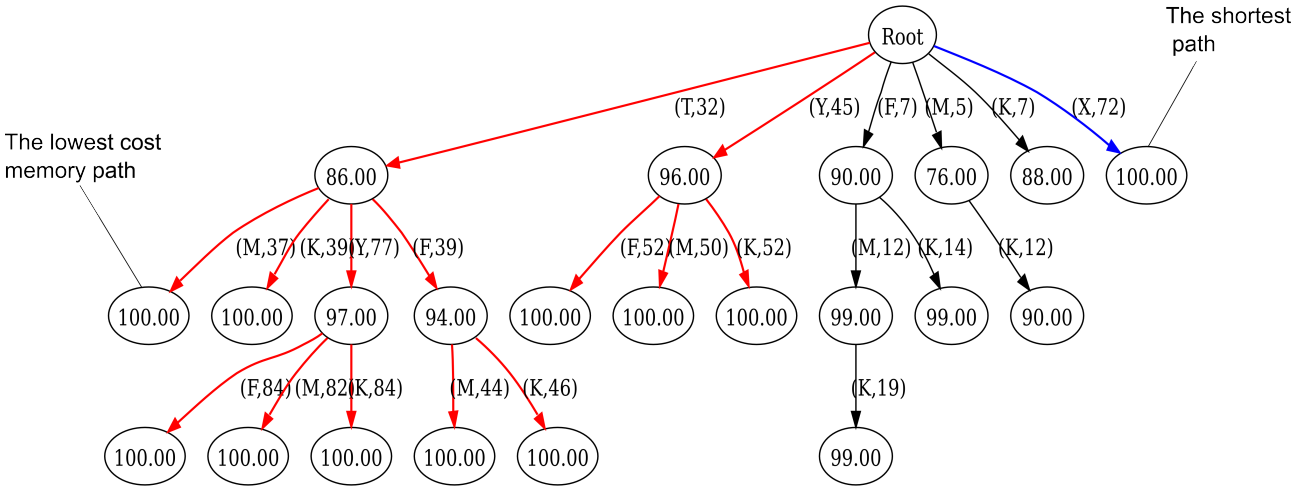



Figure 3.5: Memory cost.

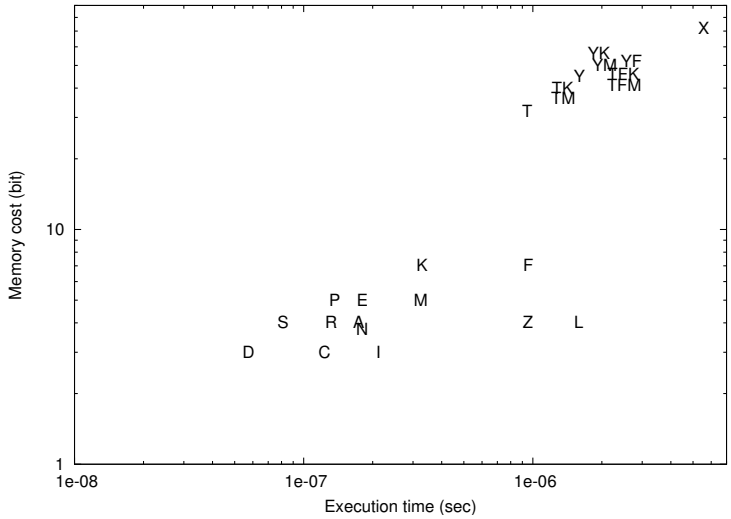



Figure 3.6: Memory cost according to execution times of criteria.

are the criteria used:

Contour-based criteria: $P$, the number of 1 having at least one neighbour at 0; $A$, the number of 1 having at least three neighbours to 0 and forming a right angle.

Region-based criteria: $S$, the number of 1 of the part; $L$, the maximum length between 1 of the part; $M$, the sum of the number of bits that change; $Y$, the product of all Manhattan distances between 1 etc. (all the criteria are presented in [31]).

We are interested by combinations reaching total differentiation, i.e. which give no error in part recognition. The results show that the minimal combinations of criteria for matrices of size $3 \times 3$ are $CC_i = \{\{TM\}, \{TK\}, \{YF\}, \{YM\}, \{YK\}, \{YE\}\}$ and for matrices of size $4 \times 4$ are $CC_i = \{\{CFIDMRZ\}, \{CFILMRZ\}\}$.

Figure 3.5 shows an example of memory consumption for all combination of the criteria $T, Y, F, M, K, X$. For example, the combination $TM$ reaches total differentiation with 37 bits.

Execution times necessary for each criterion are also measured. Figure 3.6 presents the scatter of points of memory cost function of execution time of criteria. Execution times vary greatly depending on the criterion.

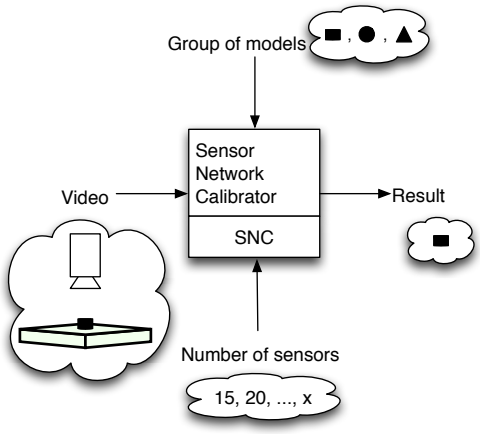



Figure 3.7: Overview of the calibrator.

### 3.2.3 Conclusions

This work presented an exhaustive framework allowing to identify the criteria reaching a total differentiation among a set of criteria. The tests on groups of 3 parts show that some combinations of two criteria for matrices of size $3 \times 3$ reach total differentiation. We have also considered the memory cost and execution time of the criteria and combinations of criteria that achieve a total differentiation.

One idea for future work is to allow a more flexible rotation (e.g. a $5^\circ$ step-by-step rotation). Another idea is to develop distributed algorithms for criteria in order to implement them in the smart surface and compare their execution time. Both ideas are considered in the next sections.

## 3.3 Sensor network calibrator

One of the parameters of the smart surface is its number of sensors. Due to space restriction, this number has to be chosen as wisely as possible: if it is overestimated, the design of the surface will be impossible, if it is underestimated, the part differentiation will be impossible. It is therefore a crucial parameter that has to be set, so we inquire about the calibration of the surface.

In the previous section we presented as future work the free rotation. In the current section we present this free rotation. Another aspect of this work is that it does not do exhaustive research, but only a few various models.

### 3.3.1 Calibrator overview

The calibrator, presented in figure 3.7, allows to define the number of sensors to be embedded into the smart surface. It receives as input:

- the video from the camera which is positioned above the surface;
- the group of models which is recognised by the surface;
- the sizes of the sensors grid to test.

The calibrator provides as output the result of the differentiation, which is the answer to the following question: what is the differentiation rate for each of the given sizes of sensors grid?

Figure 3.8 presents the structure of our calibrator. The calibrator has two stages.

**Offline stage** The goal of this stage is to find, for each group of models and each size of sensors grid, the best combination of criteria for the differentiation of the models. It has five steps:

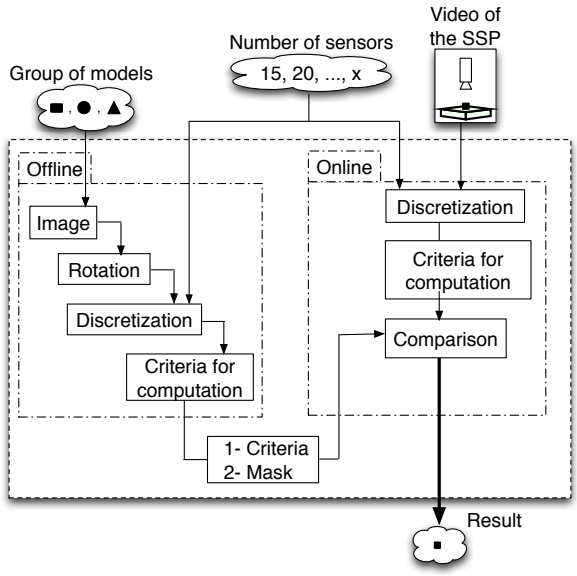



Figure 3.8: Global structure of our calibrator.

1. all rotations[1] of 1° are generated, $MRot_i$, from the image of the model $M_i$, given as input, with respect to the centre of the image, and all translations of $width(MRot)/10$ pixels are generated from the image $MRot_i$;

2. a grid corresponding to the positions of the sensors (middle point of the cell) is drawn on the images;

3. the images are discretised in a matrix by affecting 1 if the sensor is covered by the part and 0 otherwise;;

4. unique masks corresponding to the initial matrix without the rows and columns that contain only zeros are saved;

5. the values of each criterion are calculated for all masks of the model.

Among all the calculated criteria, the best criterion is chosen according to the differentiation rate, the memory cost and the execution time (cf. previous section). Finally, this criterion and its masks are given as input to the online stage.

**Online stage** In this stage, an attempt to differentiate the part on the surface is made using the differentiation algorithm, the various size sensors grid given as input and the results of the offline stage. It has four steps:

1. the camera takes a sequence of images of the surface and the part on it;

2. each image obtained is discretised (the image has only black and white pixels) to each grid size given as input to the calibrator, and the mask of the binary representation of this part is extracted;

3. the values of all combinations of the criteria of the masks are calculated;

4. the values of the criteria are compared with the values of the criteria of the models, in order to differentiate the part.

[1]The OpenCV library was used for rotations and translations.

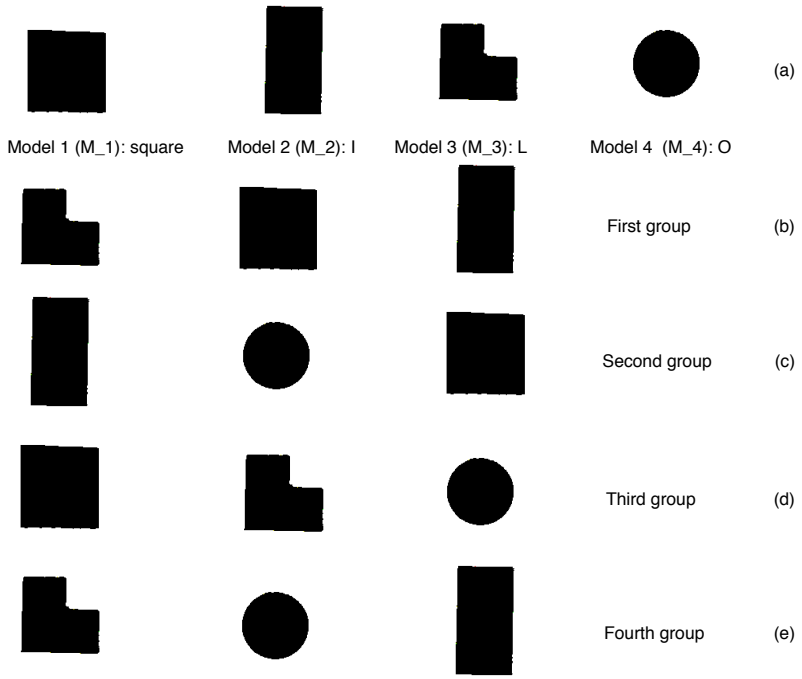



Figure 3.9: (a) Definition of our models, (b)–(e) all groups of three models.

### 3.3.2 Experimental results

The aim of our work is to parametrize the smart surface, i.e. finding the right size of the sensors grid to be used for differentiating groups of given models. For this, numerous experiments have been performed on the surface. The differentiation is made by computing the different criteria and by applying a differentiation algorithm (presented in previous section).

A set of four basic models have been chosen (see figure 3.9(a)). Starting from these models all group of three models have been generated (see figure 3.9(b)–(e)). Several sizes of grid sensors are used, as shown below.

**Offline stage** The offline algorithm is applied to each group of models. As already described, it consists of:

1. Generate all rotations of 1° and all translations of 10 pixels of the model images (size 550x550);
2. Discretize these images using the following sizes of sensors grid: (15, 15), (20, 20), ..., (50, 50);
3. Save all unique binary representations of these images;
4. Compute criteria values for each size of sensors grid.

The offline stage consists of computing the values for differentiation criteria calculated for the models $M_1, M_2, M_3$. Figure 3.10(a) represents an example of results of the offline phase, with $D, C, A$ being the differentiation criteria for the models $L, O, I$, the last group in figure 3.9(e). Figure 3.10(b) presents the tree generated from the results shown in figure 3.10(a).

**Online stage** The aim of this stage is to determine for each group of models, the size of a grid of sensors that allows differentiating our models. For this purpose, the following values must be computed for each group of models:

1: **for** each size of the sensors grid **do**
2: **for** each $M_i \subset$ models of the group **do**
3: The differentiation rate for the $M_i$
4: **end for**

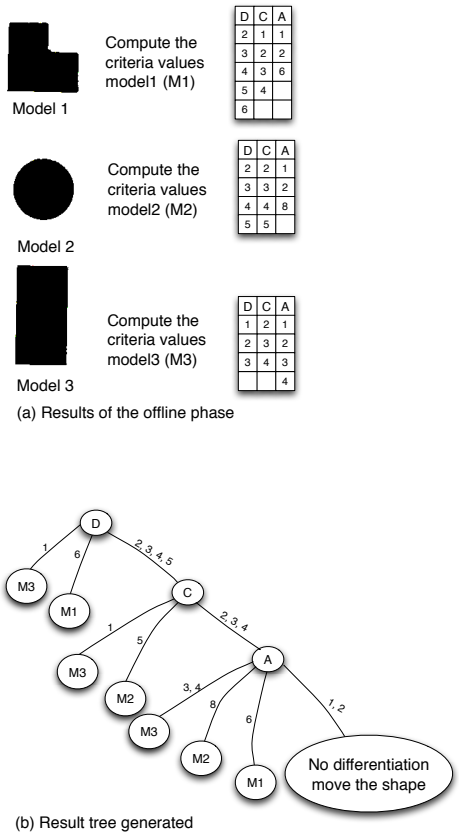



Figure 3.10: An example of the result tree of criteria values according to a combination of criteria and a set of models.

5: **end for**

The size of the sensors grid of the surface must be chosen so that it allow the best differentiation rate. For that, the average differentiation rate of our models has been computed too. Figure 3.11 presents the differentiation rate of each model in the group of models $\{L, O, I\}$ (see figure 3.9(e)), and their average, as computed with this algorithm during experiments. It can be noticed that 35x35 sensors yield a higher differentiation rate in average. Similar results are obtained for the three other groups $\{square, I, L\}$, $\{square, L, O\}$ and $\{square, I, O\}$ (presented in figures 3.9(b), 3.9(c), 3.9(d)).

Taking into account these results we can state that a sensors grid of $(35, 35)$ is an appropriate parameter for the smart surface because a high differentiation rate average is obtained.

### 3.3.3 Conclusions

This section presented a sensor network calibrator which allows to parametrize our smart surface by determining the number of sensors giving the best results in terms of differentiation rate. Tests performed on all groups of 3 out of 4 models show that a sensors grid of $(35, 35)$ is appropriate.

## 3.4 Implementation and validation on a functional platform

After finding out the best criteria for part differentiation and the most appropriate size of sensor grid, we turn our attention to implementation and validation of the algorithms produced on the functional surface, i.e. with motion control included.

### 3.4.1 Part reconstruction and differentiation

**Offline stage** The offline stage is executed before using the smart surface and can be done on a computer with a camera taking images of the models. A model is the shape of one of the objects

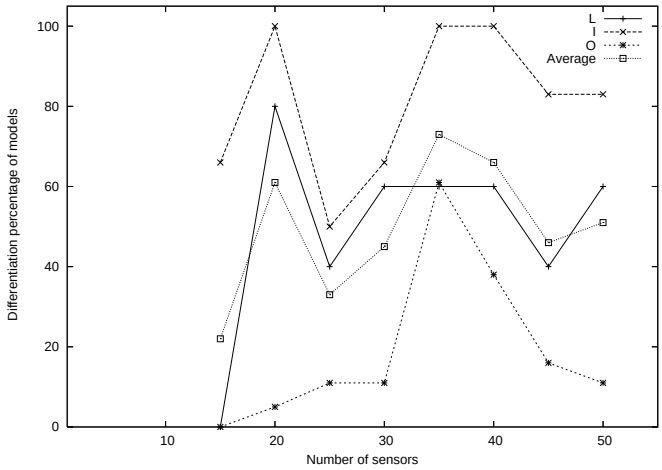



Figure 3.11: Differentiation percentage of models $L, O, I$ and their average.

(micro-parts), which can be placed on the smart surface. This stage allows to construct a database in which each model is characterised by a set of criteria values, and this database is used as input in the online stage. The method to generate this is identical to the offline stage presented in the previous section. Among all criteria, only those with a good differentiation rate, low cost memory and fast execution time (see section 3.2) are considered.

**Online stage** The aim of this stage is to differentiate, in real-time, an object on the smart surface using criteria values previously calculated in the offline stage. For that, the surface must first reconstruct the global view of the object, then apply the differentiation algorithm to differentiate the object. Thus, the online stage is divided in two phases, reconstruction phase and differentiation phase, detailed in the following.

*Reconstruction phase.* Once the object is on the smart surface, each cell knows its status (if it is covered by the object or not), but it does not know the status of its neighbours. The cells covered by the object are called *active cells*. Each active cell executes an iterative process with the aim to reconstruct the binary representation of the object. This iterative process consists in two steps: communication and computation steps.

In the communication step, all active cells are communicating with their neighbours. This communication is done by sending a message. Each active cell sends a message that consists of a matrix of bits, whose size is equal to the smart surface size (1 bit per cell). Initially, the matrix of each cell is filled with 0, except for the bit corresponding to itself, which is set to 0 if no object is seen by the sensor, and 1 if the object covers the sensor; in fact, in the first communication all the active cells know only their status and nothing about the state of their neighbours. Throughout the re-iteration of the communication phase, all the active cells will extend their state thanks to the received matrices from their neighbours, until they reach a global view of the object.

The computation step consists in updating the view of each cell according to the views collected from its neighbours in the communication step. This update consists in applying the union operator between all received matrices and itself one. Formally, let $M_{Cell}$ be a matrix corresponding to the view of cell *Cell* and $M_{1,1}$, $M_{2,1}$, $M_{3,1}$, $M_{1,2}$, $M_{3,2}$, $M_{3,1}$, $M_{3,2}$ and $M_{3,3}$ the matrices corresponding to the 8 neighbouring views respectively. The updated matrix of the cell *Cell* is obtained by applying the equation 3.3:

$$M_{Cell} = \bigcup_{i=-1,j=-1}^{i=1,j=1} M_{i,j} \tag{3.3}$$

Figure 3.12 shows, as example, the matrix of the four active cells during each computation step.

It can be proved that after $N$ steps ($N$ is the height plus the width of the object), this algorithm converges and each cell has the binary representation of the object currently on the smart surface. The number of steps is therefore not dependent on the smart surface size.

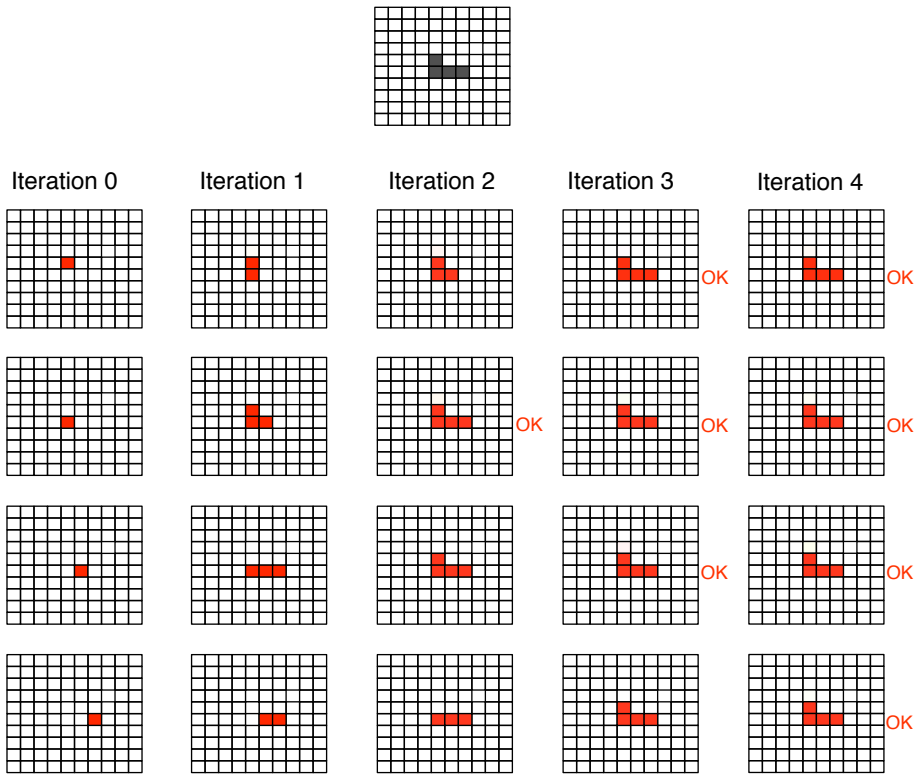



Figure 3.12: Example of the matrix of the four active cells during each computation phase.

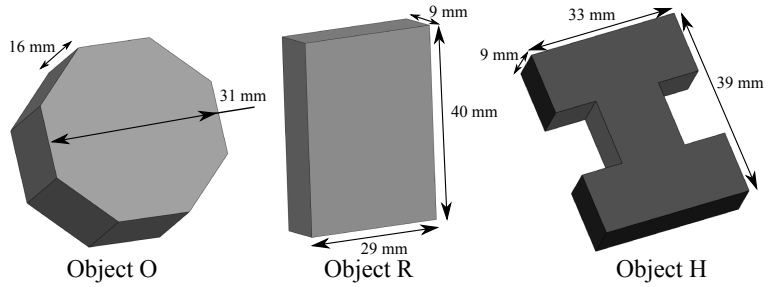



Figure 3.13: The three different objects to sort.

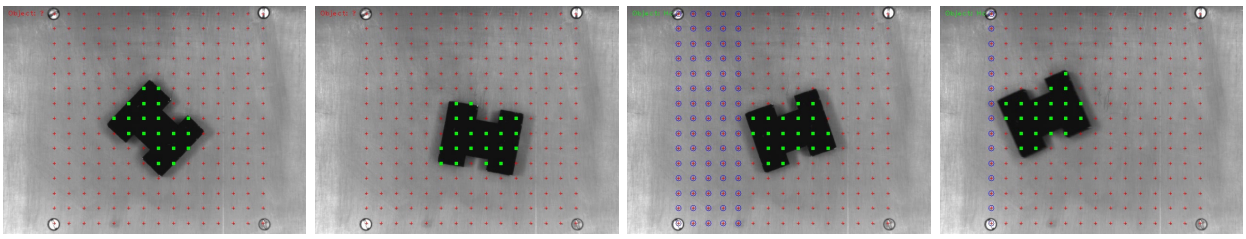
*Differentiation phase.* Once the binary representation of the object is obtained, each cell computes the criteria values of the object. These values are compared to the criteria values for each model in the database, computed in the offline stage. If there is a criterion or a combination of criteria which uniquely identifies the object (either difference of values is null, or below a specified threshold), then the object is considered to be differentiated.

The differentiation block sends to the control block the information of the differentiation result and the position of the object (coordinates of the bounding rectangle) on the smart surface. The control block takes the responsibility from now on, either to move the object to its destination (if the object has been differentiated), or to give the object a small movement in order to retry differentiation.

### 3.4.2 Experimental results

We chose a set of 3 different objects to carry out the experiments, shown in figure 3.13. The O and R objects must be sent to the east edge of the surface, the H object to the west edge. This task could fit for example in the feeding of assembly line workstations.

Figure 3.14 shows an image sequence extracted from a video made during the experimental surface. First, an object is placed on the surface. The distributed architecture maintains it in levitation by moving it in the centre of the surface until the images of the object allow to identify it. Finally, when



(a) t=0.4s, the object is not identified

(b) t=2.4s, the object is still not identified

(c) t=3.4s, the object is identified and the valves on its left are opening

(d) t=4.1s, the object is moving toward the west edge of the surface

Figure 3.14: Image sequence of an experiment with the H object (dark crosses represent inactive sensors, light squares represent active sensors and dark circles represent opened valves).

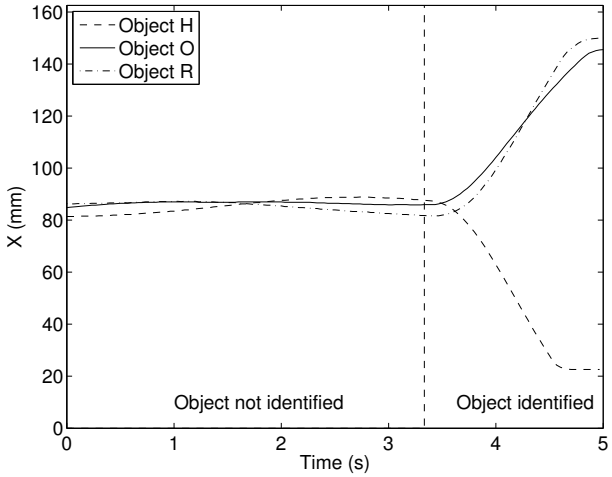



Figure 3.15: Abscissa of the center of the object according to time for three experiments with different parts.

the object is identified (H in this case), the controllers open the correct valves in order to convey the object to the desired direction. Finally, when the object is collected outside the surface, another one is placed on it and the process restarts. All this can also be seen in figure 3.15, which shows the trajectory (abscissa projection) of three different objects placed successively on the surface: It takes a few seconds to recognise it, afterwards it is moved to the appropriate destination.

In these experiments we use a set of 17 criteria sorted by their relevance for differentiation. When an object is on the surface, cells calculate the values of the first criterion and compare them to the values of the same criterion in the database. If they successfully differentiate the object (as being of type R, O or H), the process stops, otherwise it passes to the following criterion. If no criterion is successful, the object is called not differentiated *NoDiff*.

In practice however, an object can sometimes be wrongly differentiated. To cope with this, cells execute several times the complete recognition process, and use a predefined threshold (e.g. recognised at least 10 times as part H) to finally consider the object as being of that type. Note that in these experiments we aim not to obtain a fast differentiation but to correctly differentiate an object using a set of values obtained from emulated sensors.

Figure 3.16 shows the differentiation result obtained for the distributed architecture by observing several postures of the object before identifying it. The differentiation threshold is arbitrarily set to 60. In each figure we notice that, even if sometimes an object is wrongly recognised, a differentiation threshold greater that 60 gives the correct result for H, O and R.

The process is highly reproducible and robust. We performed dozens of experiments for each of the objects. It can be noticed that for each experiment, the object was successfully identified and conveyed.

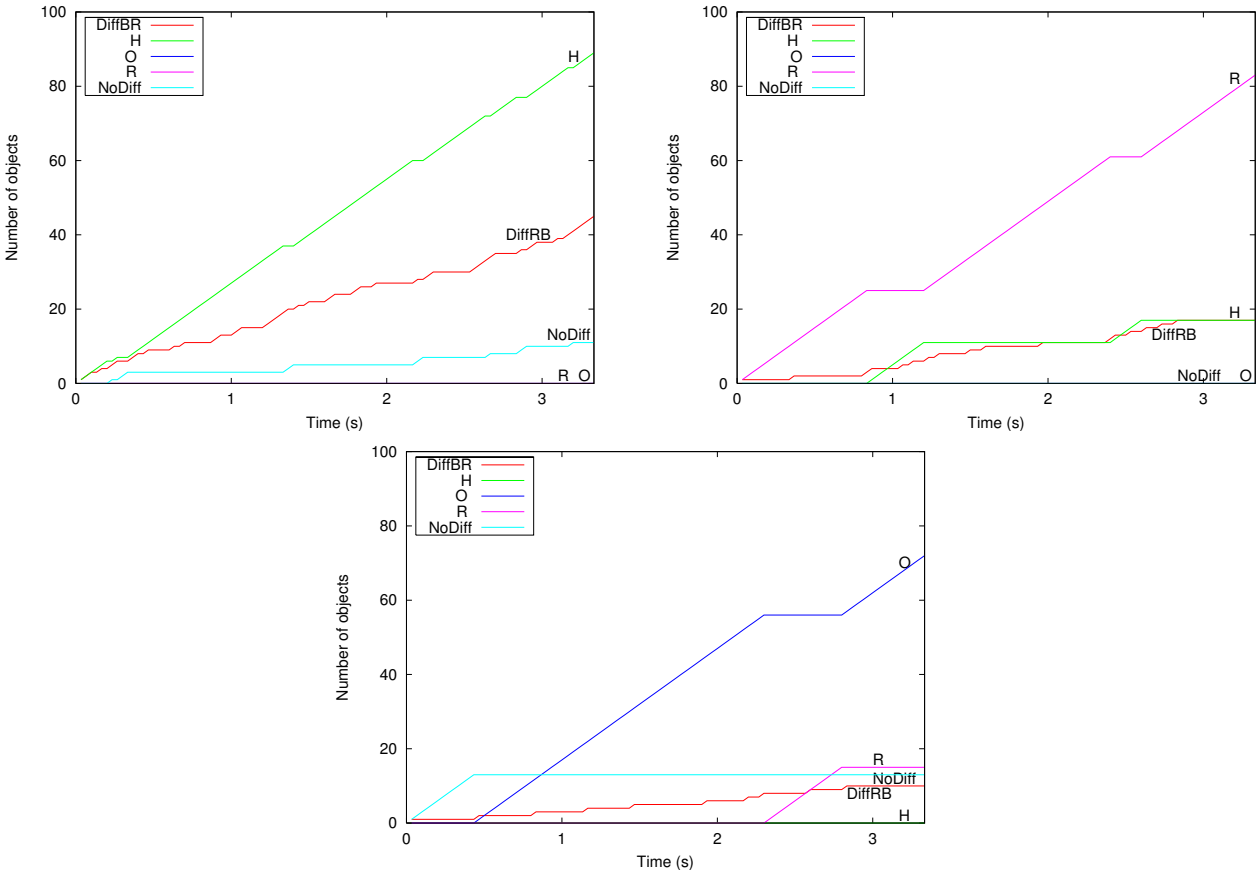



Figure 3.16: Differentiation result for experiments when the H, O, and R object respectively is put on the smart surface.

### 3.4.3 Conclusions

This section presented algorithms to reconstruct a shape based on distributed information and to recognise it. Results show that the algorithms proposed are really able to recognise the micro-parts and to convey them to the desired border of the real smart surface.

## 3.5 Enhanced part differentiation

The previous section proposed a distributed total part differentiation, which takes place after part reconstruction. In this section we propose an enhanced distributed differentiation method based on gaps.

Our papers where the enhanced algorithm is presented [66, 67] presented also a formalisation of the distributed state reconstruction phase, and a smart surface simulator to evaluate the distributed algorithms. Since these two tasks were done by another laboratory involved in the project (LAAS), I do not present them here.

### 3.5.1 Part differentiation with gaps

We first note that the value of the criteria can vary according to the orientation and position of the part on the smart surface. In the case of part rotation for example, we have observed that the value of criterion surface of a 3x3 square (where the unit of distance is the length of a cell) can vary from 9 to 13 according to the orientation of the part on the surface. In the following, we present two new approaches for part differentiation, both based on gaps.

The first approach relies on the use of a single reference position for each model. Each cell computes for each criterion the gap between measured value of the criterion for the current position of the part on the smart surface and value of the same criterion for a single reference position of the models (latter values were stored in the database during offline stage). To increase usefulness of this approach, only a subset of well known criteria like surface or perimeter of the part can be considered, whereas criteria that magnify tiny differences between parts, like product of the differences between consecutive columns and consecutive lines, are discarded. Let $q$ be the number of considered criteria, $m$ the number of models, $r_i(j)$ the reference value (computed offline) of $i$th criterion of the $j$th model, and $c_i$ the value of the $i$th criterion of a part on the smart surface. Then all cells compute concurrently and independently the following gap for the part that covers them:

$$g(j) = \frac{1}{q}\sum_{i=1}^{q}\left|\frac{r_i(j)}{c_i} - 1\right|, \quad j \in 1, \ldots, m \tag{3.4}$$

The second approach relies on the use of a set of reference positions for the models. We take into account rotations of parts. Without loss of generality and for symmetry reasons, only rotations by $1^\circ$ to $45^\circ$ are considered. Let $D = \{1, \ldots, 45\}$, $r_i^d(j)$ the reference value of $i$th criterion of the $j$th model with $d^\circ$ angle (computed offline), and $C(j)$ the set of all reference values for $j$th part $C(j) = \{(r_1^d(j), \ldots, r_q^d(j)), d \in D\}$. Then all cells compute concurrently and independently the following gap for the part that covers them:

$$g'(j) = \min_{d\in D}\left\{\frac{1}{q}\sum_{i=1}^{q}\left|\frac{r_i^d(j)}{c_i} - 1\right|\right\}, \quad j \in 1, \ldots, m \tag{3.5}$$

We note that the former gap $g$ presents the advantage of requiring limited memory and small computing time, while the latter gap $g'$ permits one to expect better part differentiation; this is particularly true in the case where parts can have any orientation on the smart surface.

The decision concerning part differentiation at each cell relies on the respective values of the gaps. The part $j$ that is chosen corresponds to the gap $g(j)$ or $g'(j)$ that is the closest to zero.

The method with gaps is particularly interesting when some cells present faults, e.g. sensor faults (giving a faulty local state) or processor faults (leading to erroneous discrete representation of a part), or when parts can be altered (i.e. slightly modified).

| Criterion | Sq (%) | I (%) | L (%) | Average |
|---|---|---|---|---|
| S | 37.0 | 52.0 | 57.3 | 48.7 |
| A | 33.5 | 40.5 | 48.5 | 40.8 |
| AS | 59.5 | 74.0 | 68.0 | 67.1 |

Table 3.2: Differentiation rates for criteria S and A with total differentiation.

| Criterion | Sq (%) | I (%) | L (%) | Average |
|---|---|---|---|---|
| S | 100 | 99.0 | 98.0 | 99.0 |
| A | 100 | 99.0 | 78.0 | 92.3 |
| AS | 100 | 100 | 96.5 | 98.8 |

Table 3.3: Differentiation rates for criteria S and A with first gap $g$.

### 3.5.2 Simulation results

The simulator written by our colleagues allows to build a smart surface with any size and create various basic parts like squares, rectangles, parts with L shape or I shape and so on, that will become models. It also permits to place the generated part everywhere on the smart surface. Shapes can be rotated on the smart surface and sensor faults can be introduced. One can also choose particular criteria and a differentiation method, i.e. a gap-based method or the total differentiation method presented in previous section. After choosing all these parameters, the multithreading simulator executes the algorithm for each cell and displays the results of the part differentiation phase, and some statistics.

We compare the gap methods and the total differentiation method presented in the previous section using the results given by the simulator.

We consider three parts: a square, referred to as the Sq part, an L shape part and an I shape part. The three parts have been randomly placed on the surface 200 times, leading to 200 draws (images) with the simulator. For each draw, the simulator computed the values of 17 criteria. The criteria were afterwards sorted according to the differentiation rate and only the criteria with the best differentiation rates have been selected. The two top criteria are S and A, where S denotes the surface and A the sum of angles of type "V" (see section 3.2).

Table 3.2 displays the results for the total differentiation method. For 200 draws, we note that the criterion S alone correctly differentiated the part Sq in 37% of cases. The combination of S and A criteria give better results: The part Sq was correctly differentiated in 59% of the cases, and the average differentiation rate increased from 40% (for A alone) and 48% (for S alone) to 67% (for AS).

Table 3.3 displays the results with gap $g$ and table 3.4 the results with gap $g'$:

- With a single criterion, e.g. S, the average differentiation rate can reach 99% with $g$ (first row of table 3.3) and 99% with $g'$ (first row of table 3.4). We recall that the average differentiation rate is 48% with the total differentiation method (first row of table 3.2).

- With a combination of criteria, the average rates are 98% (third row of table 3.3) and 92% (third row of table 3.4) for $g$ and $g'$, respectively. We recall that the average differentiation rate is 67% with the total differentiation method (last row of table 3.2).

We conclude that the gap-based differentiation methods give better results than the total differentiation method when parts can have any orientation on the smart surface. Additionally, in some cases one criterion alone may be sufficient to reach almost 100% differentiation rate.

| Criterion | Sq (%) | I (%) | L (%) | Average |
|---|---|---|---|---|
| S | 100 | 99.0 | 98.0 | 99.0 |
| A | 100 | 99.0 | 78.0 | 92.3 |
| AS | 100 | 99.5 | 79.0 | 92.8 |

Table 3.4: Differentiation rates for criteria S and A with second gap $g'$.

### 3.5.3 Conclusions

This section presented two new distributed part differentiation methods based on gaps. A simulator was used which creates the surface and models, and places randomly a model on the surface. The new differentiation methods give clearly better results than the previous differentiation method.

## 3.6 Tree-based storage

The previous section presented a new type of differentiation methods based on gaps. In this section we focus on how the database is stored in cells and what messages are exchanged between them.

Section 3.4 described the matrices created during the reconstruction phase and exchanged between neighbours. A matrix contains the state of the whole surface. The amount of data communication traffic is thus fixed and the updating method is simple. The matrix, however, has redundant information.

In the current work, we analyse a new data structure for the database which stores information about the models to be differentiated, and for the messages exchanged between neighbours. The new data structure is a tree-structured array, and, compared to matrices, seems a better way to represent knowledge. The array has a variable length and each value gives the state of the sensor at a relative position: north, west, south, and east. In the process, we also needed to update the online stage (both reconstruction and differentiation phases). Whereas in the previous approach each model was stored in the database rotated at various degrees, in the current approach each model is stored in one exemplar only. Thus the new structure avoids redundancy and frequent comparison between the object and each representations of a model. Also, a tree can be relatively easily to construct, and sub-trees can be cut off or attached to another node (distribute the database on several cells) to actually form the processes of knowledge composition and decomposition.

Representing knowledge in a tree-based structure is an idea which can be applied to other fields where processing power and storage capacity are limited. This is for example the case for ubiquitous devices [48], also known as u-objects [130, 131], like mobile devices, sensor devices, RFID devices, wireless devices [72, 104] etc., which are increasingly getting more common and smaller, even invisible to the naked eye [190]. Embedded Intelligence Entities (IE) (data + knowledge + reasoning engine) [98] in u-objects may be one theoretical approach to represent information for such devices, but at the present time they are impractical. However, Intelligence Entity Sharing (IES) from external systems is a practical alternative approach. In this approach, a u-object can request/rent/borrow whenever necessary an intelligence entity from an intelligence entity pool (IEP) to provide intelligent/smart/autonomic support or services. The intelligence entity pool needs an appropriate knowledge-representing structure, with which the intelligence entity can be flexibly composed and decomposed.

### 3.6.1 Tree creation and transformation

As already written, both phases of the online stage have to be updated to use trees.

*Reconstruction phase.* The root of a tree is the state of the cell itself, and its child nodes are the trees of its four neighbours. The state of a cell is active when an object is above its sensor, or passive when there is no object above it.

The tree generation process, leading to the reconstruction of the current object image, takes the following steps:

1. Each active cell generates its initial tree-structured array which is only composed of its current state.

2. Each active cell sends to its four neighbours its tree, removing the values corresponding to the destination (see figure 3.17).

3. Each cell receives trees from its four neighbours.

4. Each cell updates its tree by replacing its four children with the four trees received.

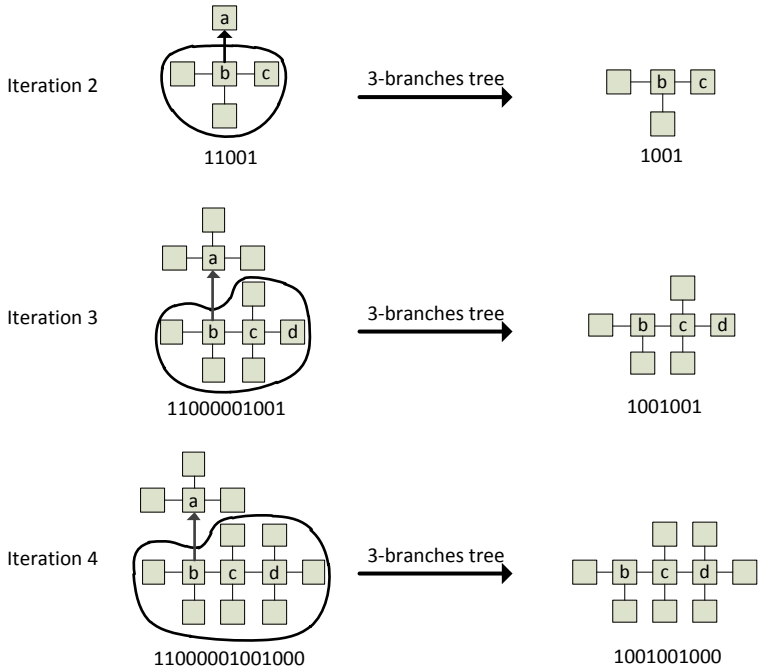



Figure 3.17: The tree-structured array of cell "b" and the three subtrees sending to cell "a" in the object reconstruction process.

5. If the leaf values of the received trees are all 0 or the trees do not contain any updates as no sensors have changed state, the reconstruction phase is over. If not the reconstruction algorithm continues with step 2.

As a result, the tree of each cell expands until the whole object is reconstructed.

As several different paths exist from one cell to another, a cycle can be created leading to an infinite growth of the trees. Such cycles are avoided through an algorithm explained in [171].

Figure 3.18 shows a complete example of the reconstruction process of an object. In this figure, each square is a cell in the smart surface, and each cell has four neighbouring cells denoted as "N", "W", "S", and "E" based on their relative positions. In this example, there are four active cells, named "a", "b", "c", "d", and each cell receives the state of the other cells in four communications. Among the four neighbours, one of them is the destination node which corresponds to the parent node, and the other three are regarded as child nodes. Therefore, all nodes except the root have three children, and a line without square represents the position of the parent node.

*Differentiation phase.* We recall that the database with the models is constructed during the offline stage. In the previous approach, a number of criteria corresponding to each model are generated and stored in the model database. Instead, in the current approach, each model is stored as one binary array which corresponds to a 4-branch tree.

To simplify the differentiation process, the root cells of all the arrays in the database have to be fixed as far north as possible, then as far west as possible. Therefore, the array generated in the reconstruction phase has to be transformed into a unique array with the most northern and western cell as root cell. This is done through a repetitive transformation, described here briefly.

The cell scans the tree-structured array from the most left. Since the first bit is always the state value of the cell itself, it starts checking the second bit. If its value is 1, it means the cell has an active north neighbouring cell; it transforms then the tree so that the north neighbouring cell become the root cell. Figure 3.19(a) illustrates how the root cell "b" is changed to its north neighbouring cell "a". The algorithm to change the array 11000001001000 to 10010010010000 is the following:

1. adding 1 to the head of the sub-array (001001000) of cell "b",
2. inserting them (1001001000) to the position (pointed by an arrow) between the 4th and 5th bits
3. and finally deleting the first bit (1)

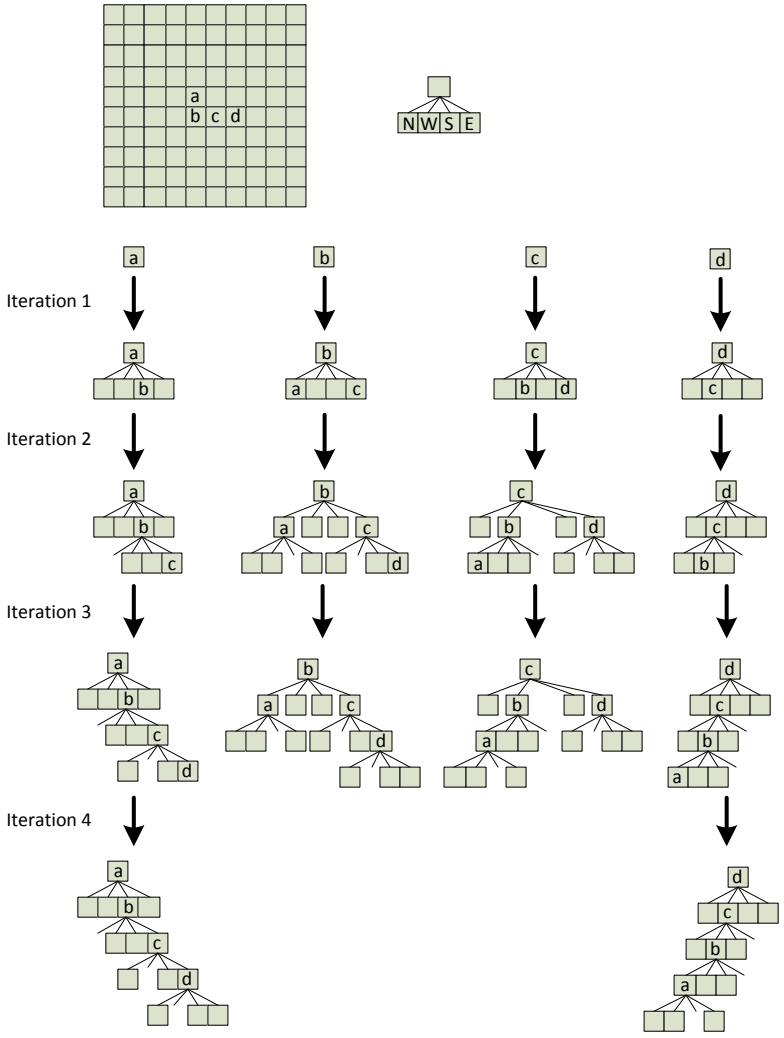



Figure 3.18: The tree evolution in each active cell during object reconstruction process.

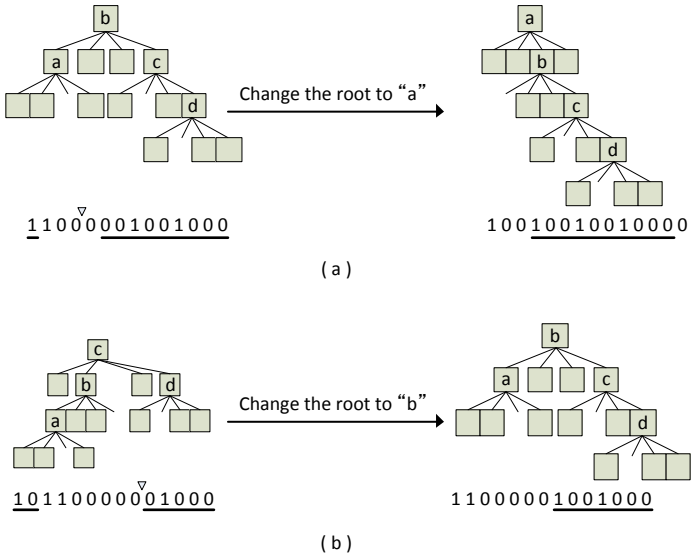



Figure 3.19: Array transformation to change the root cell.

If the second bit value is 0, no active cell is found at north. The transformation algorithm checks whether the cell has an active west neighbouring cell using a similar algorithm, shown in figure 3.19(b).

These steps are repeated until the root becomes the most northern and western cell. Then, the array is numerically compared with all the models, and based on the result the control block will send the object to the destination, or will rotate it to retry differentiation.

### 3.6.2 Numerical results

We evaluated the proposed approach in terms of the number of communication iterations, the amount of communication traffic, the length of computation time and the size of the memory footprint, and compared it with the previous approach presented in section 3.4. The greatest difference between the two approaches appears on communication traffic [171], presented in the following.

#### Communication traffic

As an example, the amount of communications in a $10 \times 10$ smart surface is calculated. The traffic amount is variable in the proposed approach, because the message consists of the states of cells known by the sender and the number of those cells increases with time. The amount of traffic for the example in figure 3.18 is as follows. In the first step, all the messages have 1 bit. In the next step, the active cells send 4 bits, because the messages contain the states of their three neighbours (without the destination node). After that we show only the amount of the messages from cell "b" to cell "a", because this amount depends on the sender and receiver. Cell "b" generates the sent message for cell "a" as shown in figure 3.17. The amounts are 4 bits, 7 bits, and 10 bits, from top to bottom. This amount needs to be multiplied by four, since trees are sent to the four neighbours. The results for the other nodes are obtained in a similar way. Summing up, the amount of data communication traffic in each active cell is $(1 + 4 + 7 + 10) \times 4 = 88$ bits, and the amount of the traffic in the network is $88 \times 4 = 352$ bits (there are 4 active cells).

In the previous approach, 100 bits of data are sent between cells for communication. Since active cells communicate with their four neighbours, each active cell sends 400 bits of data at a time. If there are four active cells as shown in figure 3.18, the cells need to communicate 4 times in order to get all active values, i.e. the amount of the traffic is $4 \times 4 \times 400 = 6400$ bits, about eighteen times greater than the proposed approach.

### 3.6.3 Conclusions

The current work aimed to improve the distributed control of the smart surface by adopting tree-structured arrays. Representing the shapes as tree-structured data has several benefits, the most notable being reduced communication traffic. Particularly, further reduction in memory footprint can be achieved by distributing the database to several cells, i.e. each cell store only a part of the database, with the disadvantage that communication traffic will increase in order to exchange information between cells about the database values.

## 3.7 Conclusions

This section presented the work done as a member of the Smart surface project. At first, we analysed criterion performance in the general case, afterwards we improved it by free rotation/translation and made it specific to the models used on the surface, and implemented it on a real surface with motion control. Experiments were successful, i.e. objects were differentiated and moved to the appropriate destination. Finally, we proposed a two gap-based methods with faster differentiation time and analysed an original tree-based storage for model database allowing a reduced communication traffic and allowing database distribution over several cells.

Even if the fabrication of the miniaturised smart surface proved difficult, the project was successful from our perspective since it led to a real and functional surface prototype, both in fabrication and usage. It also shown that a cooperation among several scientific disciplines is harder, but can succeed.

# Chapter 4

# Communication in wireless nanonetworks

## 4.1 Introduction

Since end of 2013 I inquire about communication in wireless nanonetworks. Nanotechnology is a promising technology still in its infancy. It involves objects at nanometric scale, i.e. between 1 nm and 1000 nm. I am not interested in nanomaterials, which are already being studied and fabricated, but in communication between nanodevices. A nanodevice alone has a limited communication range, limited energy, limited processing capacity etc. Networking several nanodevices improve the system being considered. Networking them means making them communicate. Currently, two communication technologies are being considered for nanodevice communication: molecular and electromagnetic. The former uses molecules to propagate information, while the latter uses the classical electromagnetic field waves. I am interested in electromagnetic type of communication.

Electromagnetic communication using nanodevices have for a long time been ignored. The reason is that a classical nanoantenna works at very high frequencies, which are useless in practice. Indeed, following antenna theory, the size of a conventional (metallic) antenna resonating at a given wavelength $\lambda$ should be at least $\lambda/2$. So an antenna of size 1 $\mu m$ (the maximum size of a nanodevice) can process signals at a maximum $2\mu$m wavelength. The frequency is $f = v/\lambda$, where $v$ is speed of propagation of wave. We consider a free space, so we will consider $v = c$, the speed of light. Therefore:

$$f = c/\lambda = \frac{3 \times 10^8 m/s}{2 \times 10^{-6} m} = 1.5 \times 10^{14} \text{ Hz} = 150 \text{ THz} \tag{4.1}$$

Communication using frequencies above 150 THz generates big propagation losses, which leads to very short transmission range, of the order of micrometers. Therefore, they cannot be used in practice. Furthermore, at so high frequencies classical antenna theory needs to be revised.

To sum up, the choices we are faced are the following:

- either use a nanoantenna working at 150 THz or more, which makes it useless for communication in practice;
- or go down to 1 THz and use antennas of 150 $\mu$m or more, so we fail to create a nanodevice and use a microdevice instead.

A third choice appeared in 2004, with the first isolation of *graphene*, a material defined as a one atom thick two-dimensional sheet. Andre Geim and Konstantin Novoselov at the University of Manchester won the Nobel Prize in Physics in 2010 "for groundbreaking experiments regarding the two-dimensional material graphene"[1]. Graphene has remarkable characteristics, such as 100 times stronger than steel at same weight and conducts very efficiently electricity[2].

[1] `http://www.nobelprize.org/nobel_prizes/physics/laureates/2010`
[2] `http://en.wikipedia.org/wiki/Graphene`

Two laboratories in the world work on antennas of graphene: BWN (Broadband Wireless Networking Lab) at Georgia Tech (United States) directed by Ian Akyldiz, and N3Cat (Nanonetworking Center in Catalunya) at Barcelona (Spain), the latter being created by the first lab too. I am in contact with a former Ph.D. student of N3Cat (Josep Miquel Jornet) who defended his Ph.D. thesis in 2013 (under Ian Akyldiz's guidance) on nanowireless communication using graphene-based antennas.

Jornet made several studies, e.g. [108, 110], and noticed that nanoantennas made of graphene allow some specific waves (surface plasmon polariton, SPP[3]) to propagate at frequencies higher than conventional antennas at the same size. The frequency allowed is 100 higher than of classical antennas. Comparing with the result given in equation 4.1, this means that a graphene-based nanoantenna (1 $\mu$m) can process signals at 1.5 THz. This solves the puzzle we faced about nanoantenna and usable frequencies.

The next step after nanometric antenna is to a physical communication layer. Given the constraints of these antennas, such as there is not enough energy to transmit continuously, Jornet conceived a specific physical protocol, similar to widely-used IR-UWB (Impulse Radio Ultra-Wide-Band) communication. In Time Spread On-Off Keying (TS-OOK) [109], each bit of data is sent at regular pace $t_s$. Bit 1 is sent using an energy pulse of duration $t_p$, and bit 0 is simply not sent (no transmission). $t_s$ should be much larger than $t_p$, e.g. 1000 times.

The next step after physical layer is network, eventually transport and application layers. Current research is stopped at this step, i.e. there is no network layer proposed for nanodevice networks. Therefore, this is an interesting topic to work on.

With regard to TCP/IP or OSI layers, my research has always focused on upper layers in the networking stack: network, especially transport, and application, with application on video transmission, as presented in previous chapters. In order to have a feeling of the new research field, my first work was to create some applications for nanonetworks, especially using video. There is however no nanoantenna built, so no experiment can be done for the moment. Therefore, I turned my attention to simulation, work presented in [56]. Afterwards, I exploited a detail in nanocommunication with TS-OOK (sending a bit 1 consumes power, but sending bit 0 does not, as described above) and proposed a method to reduce the number of bits 1 in data to be sent, allowing to reduce the energy and decrease channel interferences among others. This work is still under review [203].

## 4.2 Nanonetwork simulation

Video streaming is a growing application on the Internet, and its growing pace is not slowing down. There have been a tremendous amount of work on video streaming over the Internet but nobody has ever studied video streaming over a nano-wireless network. We think that video streaming could be a potential application for nano-wireless networks and we know that video streaming is a challenging application for networks. First, video streaming is a real-time transmission meaning that it is sensitive to delay and jitter. Second, it is often better not to retransmit losses to avoid video freezing. That is why we wonder whether nano-wireless layers need to be tuned for video streaming.

This section studies, through simulation, different scenarios of video transmission over a nanowireless network.

### 4.2.1 Introduction

According to the last Sandvine report [170], in North America, the video streaming in fixed internet connections uses about 50% of the downstream bandwidth, far before the next usage which is HTTP with about 10% of the downstream bandwidth. This trend on video streaming predominance can also be seen for the mobile access. The first usage of downstream bandwidth on mobiles is for YouTube with 17% but if we sum up all video usages (YouTube, MPEG, Netflix) we reach 31%. This trend is also true for Europe, South America and Asia but to lesser extent. This shows that video streaming has become the first usage within the Internet.

[3]SPP waves are confined electromagnetic waves coupled to the surface electric charges between a metal and a dielectric material.

The Internet of Things (IoT) [8] federates the things that need to communicate and their requirements are different from traditional computers and humans. Some things will still need a high bandwidth and a low latency but most of them only need low power communications and, low bandwidth and high latency are not an issue. IoT is therefore growing both inside and aside from the Internet creating a new way to communicate.

Advances in micro-electro-mechanical-systems (MEMS) are enabling the design and fabrication of distributed intelligent MEMS (DiMEMS) [30]. A node in a DiMEMS system is basically composed of actuators, sensors, a processing unit and communication capabilities, which makes it a micro-robot integrated in a much larger ensemble, the IoT.

The appearance of nano-electromagnetic communications, referred to as nano-wireless in the rest of this section, changed the perspectives for communicating between small things, making possible communications between miniaturised elements, with the internet of micro-things [28], or even smaller elements, with the internet of nano-things [107]. Nano-wireless communications have been first proposed in [5], later detailed in [6] using graphene nano-antennas and been used in some micro-robot applications [25]. These antennas are very small and are able to radiate in the terahertz band.

Video streaming is therefore possible between very small things but remains complex due to the nature of Terahertz band. Propositions have been made for defining an energy-efficient physical layer [189] but transmitting video on top of these different layers is still an open issue (but has been studied in ultra-wide band networks in [36]). In fact, we think that a cross-layer approach is needed in the design of the communication layers of nano-wireless communications. This means that applications have to be proposed and tested. In a previous work [24], we have studied the integration of wireless capabilities in micro-robots of the Claytronics project, showing the enhancement created by wireless communications, but not using nano-wireless communications yet. Also, video streaming at small scale is of importance in many fields, e.g. biological, defense and security, under the assumption that tiny cameras could eventually be embedded in the environment (wall, body etc.)

This work is at the conjunction of these four research fields: micro-robots, IoT, nano-wireless communications and video streaming, and presents a preliminary study on the possibility to stream video between micro-robots viewed as IoT elements and using nano-wireless communications. The objective is to learn some lessons on the efficiency of the current nano-wireless physical layer in order to enhance it afterwards. We use two NS3 plugins: Nano-Sim, to simulate the nano-wireless physical layer, and Quality-of-Experience (QoE) Monitor, allowing to stream real video sequences inside NS3 and to evaluate the result in terms of video quality.

### 4.2.2 Background

We noticed that the only way to simulate nanonetworks is to use NS3 simulator and an external plugin (module), Nano-Sim; a video transmission external plugin also exist, QoE Monitor. NS3 [141] is the follow-up of NS2 [140], the classical network simulator used by researchers in computer networks. NS3 is supported by an active community that works on many topics (groups), and researchers can validate their contribution by comparing the existing ones. In a nutshell, NS3 uses a discrete-event model and simulates numerous network devices, protocols, applications at all OSI levels, and provides utilities to visualise results.

Nano-Sim[4] [148] was written at Technical University of Bari in Italy and simulates very roughly a nanonetwork. It has been used to test health care applications and it comprises three types of WNSN devices: nanonodes (sensors), nanorouters (with larger capabilities than nanonodes, and can route information) and nanointerfaces (large capabilities, can process data and act as a gateway to another network). It implements a simple application layer (Constant Bit Rate CBR), network layer with flooding and random routing protocols, MAC layer and physical layer (using Terahertz spectrum using TS-OOK modulation [6]).

Quality of Experience (QoE) Monitor[5] [168] was written at University of Modena and Reggio Emilia in Italy and features an H264-encoded video file reader, transmit it using RTP protocol through NS3, and a valid video writer which reconstructs a valid video file from the received data. It also computes

[4]Downloaded from `http://telematics.poliba.it/index.php/en/nano-sim`

[5]Downloaded from `http://sourceforge.net/projects/ns3qoemonitor`

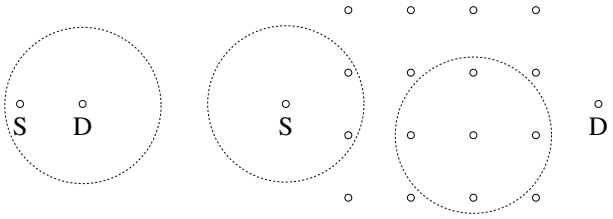



Figure 4.1: The two nanonetwork topologies used in simulation: on the left side the 2-node topology, on the right side the 18-node topology.

PSNR and SSIM metrics based on the differences between the original video at the transmitter side and the reconstructed video at the receiver side. When data is lost on the network, entire frames can be lost too. This fools PSNR and SSIM metrics, since they work on analogous frames in original and received videos. In order to have the same amount of video frames and yield a valid video data, in QoE Monitor the receiver reconstructs a valid video by replacing all the lost packets with dummy data.

### 4.2.3 Simulation setup

The two modules did not work out of the box. We used an NS3 version (3.16) which worked with QoE monitor. A first modification was to make QoE monitor work with recent version of libav (a fork of ffmpeg), which is known to change often its API. A second and most difficult challenge was to make Nano-Sim and QoE monitor work together. In both modules, packet sending is done deep inside the module. Our solution was to hack QoE code to replace QoE packet sending with calls to Nano-Sim packet sending. The source code solving these issues is freely available on Internet[6].

Unfortunately, we met limitations, simplifications and bugs in the two modules unsolved for the moment, for example:

- bug: Nano-Sim does reordering of packets whereas it should not. For example, in a simple network with two nodes (source and destination), sometimes a packet B arrives before a packet A which was sent before B.

- limitation: QoE monitor receiver discards a fragment if the previous fragment has not been received, otherwise said packet reordering leads to packet loss. As such, all packets arriving in disorder are replaced by null data in the received data. Real video clients reorder received packets instead. The end result is that the quality of received video in QoE monitor is less than in reality.

- simplification: Nano-Sim has a very simplistic propagation model (all or nothing), where packets are received if they are inside a circle of some radius from the sender, or lost otherwise.

For all these reasons, we consider our work as a rough but first study on video transmission over nanonetworks.

We used two network topologies for the tests, shown in figure 4.1. The first has two nodes, and is used to check the simulator with the two modules (QoE monitor and Nano-Sim). The second has one source, one destination and 16 relays, and is used to discover how communication is done in a multi-hop network. All the nodes are motionless. The distance between two consecutive nodes is 1 cm. The communication range for all nodes is set to 1.2 cm, and was chosen so that the network exhibit a connectivity of 4 neighbours, and that there are several hops (more precisely 5 hops) between the sender and the destination, and contention in the network during the communication. The molecular communication is used, with flooding routing protocol and TS-OOK modulation. The pulse duration is 100 fs and the pulse interval is 10000 fs, i.e. 100 times greater than pulse duration. The transmitting power is 1000 fW.

[6] `http://eugen.dedu.free.fr`

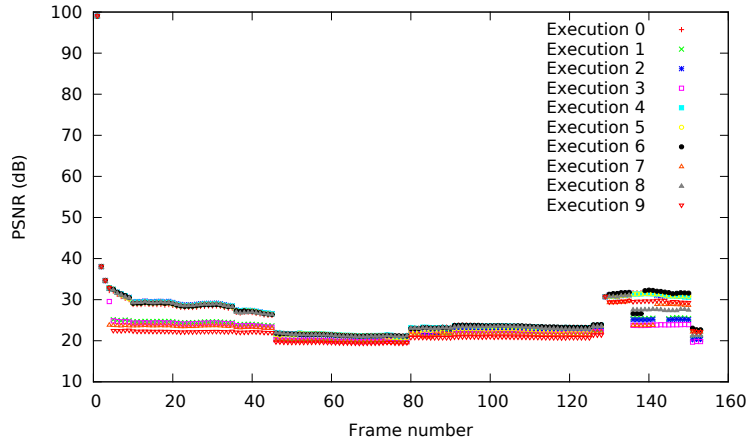



Figure 4.2: The PSNR for 2-nodes network.

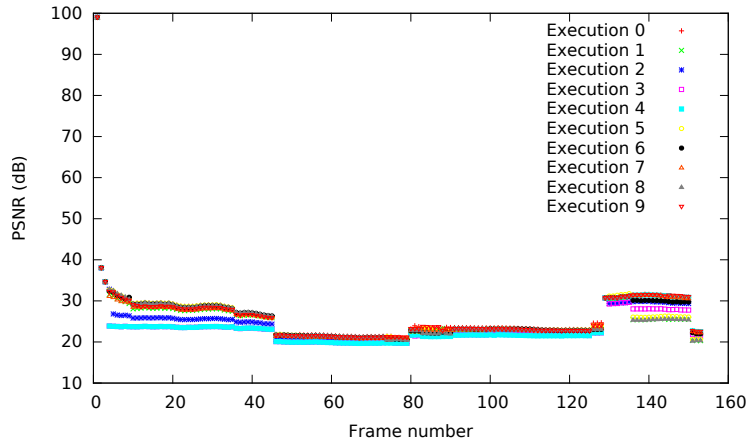



Figure 4.3: The PSNR for 18-nodes network.

There is one flow in the network. The video file used as input is the classical "news" sequence in CIF resolution. The file starts to be sent at second 2. The simulation ends when the file streaming finishes. We executed ten times each of the two topologies.

### 4.2.4 Simulation results

The PSNR metric between the received video and the sent video for 2-nodes network is presented in figure 4.2. It can be seen that all the executions give similar results. Also, the PSNR has a relatively low value (20 to 35 dB) and is quite regular, knowing that 20 to 25 dB are considered to be acceptable values for wireless transmission quality loss. No packet is lost on the network; instead the reordering done by Nano-Sim, as presented in previous section, makes QoE monitor drop packets at receiver. The abrupt changes in PSNR plot, appearing at frames 45, 80 and 130, correspond to abrupt scene changes in video file. The PSNR for 18-nodes network, given in figure 4.3, is similar to the one for 2-nodes and exhibits the same properties.

The SSIM metric for 18-nodes network is presented in figure 4.4. All the executions give similar values. SSIM curve varies much more than PSNR curve. As for PSNR, SSIM curve varies more at abrupt scene changes, but it is less visible, except for frame 130. The SSIM curve for 2-nodes network is similar to 18-nodes network.

For video transmission, another important parameter is how the packet delay changes, because it fixes the receiver buffer size. The jitter (the difference between packet delays) is presented in figure 4.5. It shows that the jitter varies generally between 30 ns and 70 ns. These values are 3 orders of magnitude lower than what is currently found on Internet, which are of order of tens of ms. As a consequence, the buffers at receiver side could potentially be very smaller than the ones on Internet. However, more

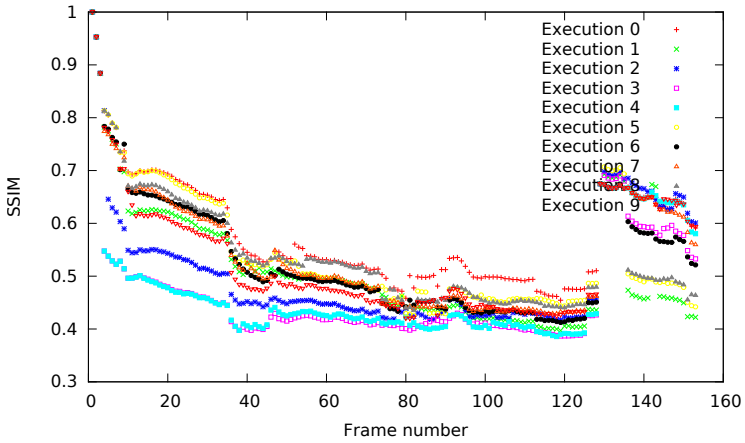



Figure 4.4: The SSIM for 18-nodes network.

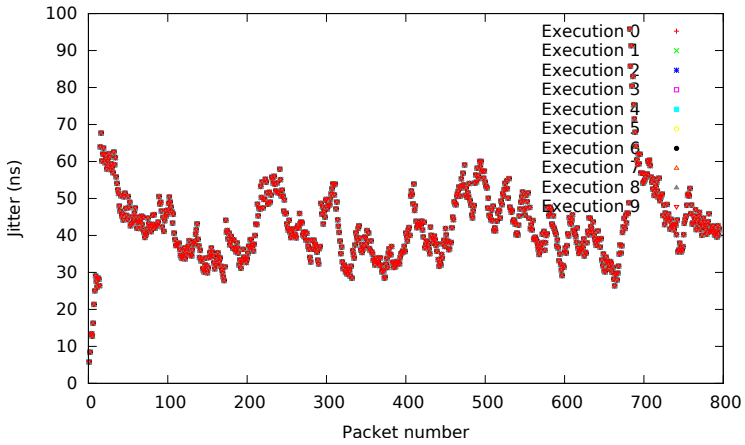



Figure 4.5: The jitter for 2-nodes and 18-nodes networks.

importantly, the figure shows that the jitter is identical for all executions, either 2-nodes or 18-nodes. This is an unrealistic result, since in reality the delay and the jitter do depend on the number of hops between sender and receiver (1 hop in 2-nodes, and 5 hops in 18-nodes network). This result shows the limits of Nano-Sim module for video transmission.

### 4.2.5 Conclusions

This section presented a study on video transmission over nanowireless networks. We tried to do experiments, but no physical device exists for that. The only possibility to study it is through simulation, using the NS3 well-known network simulator and the QoE monitor and Nano-Sim external modules. PSNR and SSIM quality metrics of the received video, together with jitter graphs have been presented using a 2-nodes and a multi-hop 18-nodes network.

This study is the first to tackle video transmission over nanonetworks. It showed the limitation of the current tools and their models. For example, the jitter results of Nano-Sim and the losses at receiver side in case of unordered packets are not realistic. The main result is that the research in this field needs better models and tools.

## 4.3 Nanonetwork energy efficiency

### 4.3.1 Introduction

In this section we introduce a source coding algorithm for nanosensor networks whose goal is to encode the data to be transmitted in order to increase the number of bits 0 in detriment to 1s. Data is divided

in fixed size symbols. In a dictionary, more often used input symbols are mapped to output symbols with fewer bits 1. Before transmitting data, sender replaces input symbols with output symbols, and the receiver replaces received symbols back to original symbols. In the process, sender takes also into account the distance between 1s in output symbols. Depending on input data, bit 1 reduction can go up to 100%. We also analyse its energy efficiency and transmission robustness using several types of real files.

### 4.3.2 Related work

Compression techniques are usually used to reduce the redundancy in the information. A classical compression technique is Huffman coding, where the most often used symbols are encoded with fewer bits [154]. Our proposed method is a variation of Huffman coding. In both algorithms, symbols are ordered according to their frequency on input data. However, whereas in original Huffman algorithm the more frequent symbols have fewer bits, in our method the more frequent symbols have the same number of bits, but fewer number of 1s.

There are many variations of Huffman code. Abrahams [1] gives a comprehensive list. She considers fixed-to-variable, and variable-to-fixed source coding. Our method is fixed-to-fixed. Input data can be infinite, can have lexicographic constraints (Hu-Tucker problem), the codeword length can have constraints, coding can have unequal cost code symbols (Karp problem). Our method is similar to the latter variant, Karp problem. Whereas in classical problems the cost of a symbol is the number of its bits, in Karp problem the cost of a symbol depends on its bit *values*, i.e. the cost of symbol $i$ is $c(0)M(i) + c(1)N(i)$, with $M$ the number of bits 0 in symbol $i$, $N$ the number of bits 1, and $c(0)$ and/or $c(1)$ different than 1.

More specifically, there have been some works in energy efficient coding to reduce the frequency of occurrences of bits 1 that can be used in wireless networks. Erin et al. [69] proposed Minimum Energy (ME) coding to transmit more frequent symbols using fewer bits 1 in wireless communication. Our method is similar, with some differences due to characteristics of nanosensor networks. For example, codewords with the same number of bits 1 are not ordered in Erin's method, whereas in our method they are ordered according to the distance between 1s. This is because in nanosensor networks it is preferred to have greater distance between adjacent 1s, for two reasons. First, to have more relaxed constraints for energy harvesting, since a larger distance between adjacent 1s gives the sensor more time to harvest the energy for the next pulse transmission [189]. Second, to have a more relaxed (less activity) channel with respect to absorption noise in high traffic; for example, it is better to transmit 1010 sequence instead of 1100, because in the first case when sending the second bit the noise from the first bit transmission still exists in the channel. Erin's article is more theoretical, whereas ours presents numerical results with different values of symbol size on various types of input data.

Prakash et al. [153] use the small part of ME coding by encoding $m$ bits into $n$ bits ($n > m$) with weight 1. This method requires more bandwidth since the size of output is bigger than input. Chi et al. [43] extend Prakash's method using larger $m$ and $n$, and provide more numerical proofs. Chi et al. [44] use variable length code with minimum average code weight.

Kim et al. [113] propose a Modified Minimum Energy (MME) coding where codewords are divided in several subframes and a bit is added in front of each subframe: 1 means no high bits in the subframe, and 0 means at least one high bit in the subframe. The purpose of subframes is to allow the receiver to sleep for the duration of subframes with only 0s, thus reducing energy on receiver. This gives good results only in specific types of data, where adding a bit of data still decreases the energy; moreover, it increases data size.

ME and MME codings are analysed in the context of Coded Division Multiple Access (CDMA) wireless sensor networks (WSN) [75]. Contrary to this, our paper proposes a variant of ME which takes into account some characteristics of wireless nanonetworks.

| Symbol | Frequency |
|---|---|
| $a_i(1)$ | $n(1)$ |
| $a_i(2)$ | $n(2)$ |
| $\vdots$ | $\vdots$ |
| $a_i(M)$ | $n(M)$ |

Table 4.1: Step 2 in NME coding.

### 4.3.3 Nanonetwork minimum energy coding

#### NME algorithm

Nanonetworks based on electromagnetic communication using TS-OOK modulation in Terahertz band have large bandwidth. The limitation in such networks are energy, computational complexity, and transmission range. Networking nanosensors in multi-hop fashion allows to reduce the computational complexity and to increase the transmission range. In TS-OOK modulation, 1 bits are transmitted with a femtosecond-long Gaussian pulse, with total energy 0.1 aJ [109], while 0s are transmitted as silence (no transmitted signal).

In this paper, we propose Nanonetwork Minimum Energy coding to reduce the energy usage for communication between nanosensors. It is a simple algorithm, suitable to the small power available in nanosensors. Data is transmitted from sender to receiver(s) as bits 0 and 1, and received as bits 0 and 1. The idea is to transmit the most often used blocks of bits with fewer 1s, in order to decrease the energy used to send the data. The algorithm for nanosensor networks is the following:

1. Segment the binary input sequence into blocks (symbols) of $n$ bits.
2. Create a table of used symbols and their frequency.
3. Create another table by sorting the symbols in decreasing order of their occurrence level, and then encode more often used symbols with fewer 1s. Output symbols with the same weight are sorted in decreasing order of the largest distance between consecutive 1s in the output symbol.

If the order of the output symbols with the same weight is not taken into account, NME coding is the same as ME coding. For example, the available 4-bit symbols with 2 bits 1 are the following: 0011, 0101, 0110, 1001, 1010, 1100. ME coding orders them in ascending order, like previously written. Instead, NME orders them in descending order of the distance among the bits 1: 1001, 1010, 0101, 0110, 1100, 0011. Thus, more often used symbols are encoded with more spaces between 1s, which is more suitable to nanonetworks, as stated before.

Note that the output of NME algorithm has the same number of total bits as the input. The only difference is the number of 1s, the output of NME algorithm having less number of 1s than the input.

In the following, we will detail the algorithm. In step 1, the binary input sequences are segmented into blocks of $n$ bits, afterwards the binary sequence is converted into symbols of $A = a_1, a_2, \ldots, a_N$. $A$ denotes the set of possible output from the random variable X. The probability mass function is denoted by $P_i = P(X = a_i)$ for $i = 1, 2, ..., N$ (where $N = 2^n$). In practice, not all the symbols are used in transmission process. In this case, the set of used symbols can be defined as $Au = a_1, a_2, \ldots, a_M$, where $M \leq N$.

In step 2, the table of used symbols and their frequency (number of occurrences) is created, as shown in table 4.1. This table allows to count the number of 1 bits. The total weight (the number of 1s) for original data is:

$$N_{original} = \sum_{i=1}^{M} n(i) \times w(a_i) \tag{4.2}$$

where $n(i)$ is the number of occurrences of symbol $i$, and $w(a_i)$ is the Hamming weight of symbol $i$ (the number of 1s in symbol $i$) [178].

In step 3, a new table (the dictionary) is created from the previous table by sorting the symbols based on their frequency of occurrence, as shown in table 4.2. More often used symbols appear upper

| Input symbols $A_i$ | Frequency | Output symbols $A_o$ |
|---|---|---|
| $a_i(1)$ | $n(1)$ | $a_o(1)$ |
| $a_i(2)$ | $n(2)$ | $a_o(2)$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $a_i(M)$ | $n(M)$ | $a_o(M)$ |

Table 4.2: Step 3 in NME coding.

in this table, i.e. $n(i) \geq n(j)$ for $i < j$. The dictionary is created before the transmission and it does not change afterwards. The total weight of NME output is:

$$N_{NME} = \sum_{i=0}^{M} n(i).w(a_o(i)) \tag{4.3}$$

### NME properties

**Definitions** An $n$-bit symbol is a string of $n$ bits. A mapping is a correspondence table between a symbol and its encoded symbol, so a function $f : S_n \rightarrow S_n$, where $S_n$ is the set of all $n$-bit symbols. A mapping is injective.

The following is an example of a 2-bit mapping:

$$\begin{aligned} 00 &\rightarrow 01 \\ 01 &\rightarrow 11 \\ 10 &\rightarrow 10 \\ 11 &\rightarrow 00 \end{aligned}$$

It has four symbols of 2 bits each, both in input and output.

**Theorem.** The number of possible $n$-bit mappings is $2^n!$.

**Examples.** There are $2! = 2$ possible 1-bit mappings, $4! = 24$ possible 2-bit mappings, and $8! = 40320$ possible 3-bit mappings.

**Proof.** The proof can be found at [203].

**Theorem.** The set of $n$-bit mappings is a subset of $2n$-bit mappings set $\Rightarrow$ The best $2n$-bit mapping is equal or better than the best $n$-bit mapping.

**Proof.** The proof can be found at [203].

**Dictionary length** In the general case, the dictionary (coding table) too should be transmitted to receiver, before the data. So the dictionary length is an important parameter to consider. However, as the data size increases, the dictionary length becomes less and less important compared to data.

The biggest dictionary for an $n$-bit mapping contains all the possible $n$-bit symbols, i.e. $2^n$ symbols. Each symbol has $n$ bits. Therefore, the dictionary contains $n2^n$ bits for input, and $n2^n$ bits for output, which gives $2n2^n$ bits. However, the dictionary can be sorted by the output symbols, so that only the input be transmitted. So the Maximum Dictionary Length is $MDL = n2^n$.

In practice, not all the $2^n$ symbols are found in the data, but only $M$, with $M \leq 2^n$. In this case, the Used Dictionary Length is $UDL = Mn$.

### Metrics to evaluate NME

**Energy efficiency** The first metric we use to measure the coding improvement is the energy efficiency $\xi$. This is the most critical parameter for nanosensor networks, since the battery capacity in nanosensors is very limited [43]. During a transmission, both sender and receiver consume energy. On the sender side, the energy to transmit symbol $i$ is $E_i = w_i.E_p$, where $E_p$ is energy of a pulse in TS-OOK modulation. Current nano-transceivers are able to transmit a pulse with total energy 0.1 aJ [109]. The total energy required for the transmission can be obtained by multiplying the number of transmitted pulse and the energy per pulse. The energy to transmit a symbol is equal to

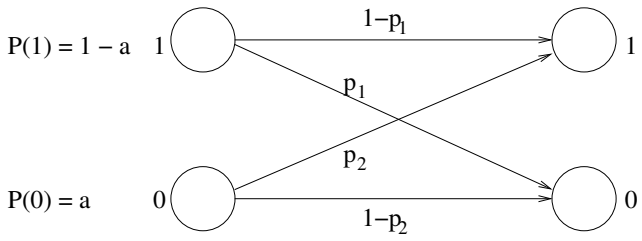



Figure 4.6: Nanonetworks use the BAC channel model.

the weight of the symbol multiplied by energy per pulse. By taking into account the frequency of occurrences of each symbol, the total energy for uncoded data is:

$$E_{original} = \sum_{i=1}^{M} E_{a_i(i)} \times n(i) = N_{original} \times E_i \tag{4.4}$$

where $E_{a_i(i)}$ is the energy of input symbol $i$, $n(i)$ is frequency of occurrences of symbol $i$, $N_{original}$ is total number of 1s in the transmitted data, and $E_i$ is energy per pulse.

NME coding aims to reduce the number of 1s in uncoded data. After performing the coding, two components need to be transmitted: data and dictionary. Then the total energy after NME coding can be described by:

$$E_{NME} = \sum_{i=1}^{M} E_{a_o(i)} \times n(i) + \sum_{i=1}^{M} E_{a_i(i)} \tag{4.5}$$

where $M$ is number of used symbols, $E_{a_o(i)}$ is the energy to transmit output symbol $i$, $n_i$ is its frequency, while $E_{a_i(i)}$ is the energy to transmit symbol $i$ the dictionary.

The energy efficiency is given by the percentage of energy reduction using NME compared to the uncoded data:

$$\xi = \frac{E_{original} - E_{NME}}{E_{original}} \times 100\% \tag{4.6}$$

A positive value of $\xi$ means that NME algorithm effectively reduces the power consumption in transmission process. However, if dictionary size is important compared to data size, $\xi$ could be negative (NME coding requires more power than uncoded transmission).

As it was already written, the receiver too consumes energy. We consider that it consumes the same power when receiving a bit 0 or 1. Thus the only difference between uncoded and NME transmissions is the transmission of the dictionary. Given that the dictionary length is normally much smaller than data size, and that the receiver consumes much less power than the sender (e.g. 10% of the power used by sender, mainly because there is no power amplifier at receiver), in this paper we only concentrate on the energy used by the source.

**Robustness during transmission** Like in any wireless channel, wireless nanonetworks are prone to errors, with specific causes: molecular absorption, spreading loss, distances greater than e.g. tens of centimetres and so on. We use the Binary Asymmetric Channel (BAC) channel, shown in figure 4.6, suitable to model nanonetwork losses [106]. In BAC model, 1 is correctly received with probability $1-p_1$ and received with error with probability $p_1$. 0 is changed to 1 with probability $p_2$, with $p_2 \ll p_1$, because it corresponds to background noise. Input bit 0 comes with probability $P(0)$ and bit 1 with probability $P(1)$, with $P(0) + P(1) = 1$.

*Codeword error rate:* Since NME uses a dictionary table and the receiver encodes the symbol received back to the original symbol, a relevant metric for coding robustness is codeword (symbol) error rate (CER) of the channel, as opposed to individual bit error rate (BER) presented in figure 4.6. $CER$ computing is taken from [106]. The probability of bit error $P_e$ can be calculated as follows:

$$P_e = p_2 P(0) + p_1 P(1) \tag{4.7}$$

where the variables involved are the probabilities shown in figure 4.6. A codeword in NME consists of $n$ bits. If a bit in the codeword is wrong, the whole codeword is wrong. The probability of correct codeword (no bit error in the received codeword) is calculated by:

$$P_c = (1 - P_e)^n \tag{4.8}$$

where $n$ is the size of codeword. Assuming bit errors are not correlated, the CER is given by:

$$CER = 1 - P_c \tag{4.9}$$

hence:

$$CER = 1 - (1 - p_2 P(0) - p_1 P(1))^n \tag{4.10}$$

where $n$ is codeword length and $p$ variables from figure 4.6.

*Application to multimedia transmission:* Until now we dealt with errors at channel level. Now we are interested in the upper level (NME) when such an error occurred. Since the decoding process in NME algorithm converts the output back into the input using the dictionary, the reconstructed symbol could be further distorted. For example, suppose that table 4.2 contains the following two lines: 1010 $\rightarrow$ 0001 and 0101 $\rightarrow$ 1001. Suppose that sender wants to send message 1010. NME encodes it to 0001 before sending it through the channel. Due to a 1 bit transmission error, the message is received as 1001, and the receiver encodes it back to 0101. Summing up, symbol 1010 is received as 0101, which means that a 1 bit error during transmission produced a 4 bits error at receiver.

In the following we use a practical example. As already presented in introduction, nanonetworks have applications in image transmission. The BAC channel create errors, so the received image could be different than the original one. To measure the quality of reconstructed image we use the classical Peak Signal to Noise Ratio (PSNR) metric, where the larger the PSNR value, the better the received image (closer to sent image) [177].

### 4.3.4 Numerical results

We use MATLAB to obtain results.

#### Energy efficiency

We first generate random binary sequences of 10,000 bits with various frequencies of bit 1. Using equation (4.6), the energy efficiency of NME on data alone (when dictionary is known by receiver) is shown in figure 4.7. The main result of the figure is that for a fixed input sequence size, the greater $n$, the greater the improvement, as expected. Also, when the input sequence has only 0 bits, the output is also all 0, so there is no coding improvement (0%); as the probability of 1 increases, the symbols with larger weight occur more often, and the coding improvement increases.

To evaluate NME more realistically, we do tests with several representative types of files: compressed and raw image/video, and program file. For instance, video files are bigger than image files; compressed video/image files have very few redundancy (this is the purpose of compression), so the number of bits 0 and 1 is about 50% each; a program file has many bytes 0. The particular files used in each category (`news_cif`, `bus_qcif` etc.) were chosen at random, without any specific reason. In the following, we detail the results of only one type of file, and give a summary for the other files. The results consists of the energy used to send the data (the number of 1s in the output of NME), the dictionary (the number of 1s in dictionary) and their sum. The energy efficiency for NME coding is measured using equation (4.6).

*Compressed image*: `lena.jpg`. Transmitted bits: 272,016 bits = 33.2 kB. The number of 1s for original is 132,740 bits. The NME performance is shown in table 4.3. The largest improvement is achieved using NME 24 bit with 36.08% energy efficiency. Note that a negative energy efficiency appears for 16 bits, because the dictionary size (26.7 kB) is comparable to data size (129 kB).

*Uncompressed image*: `lena.bmp`. NME gains in all cases, and the largest improvement is achieved using NME 24 bit with 34.22% energy efficiency.

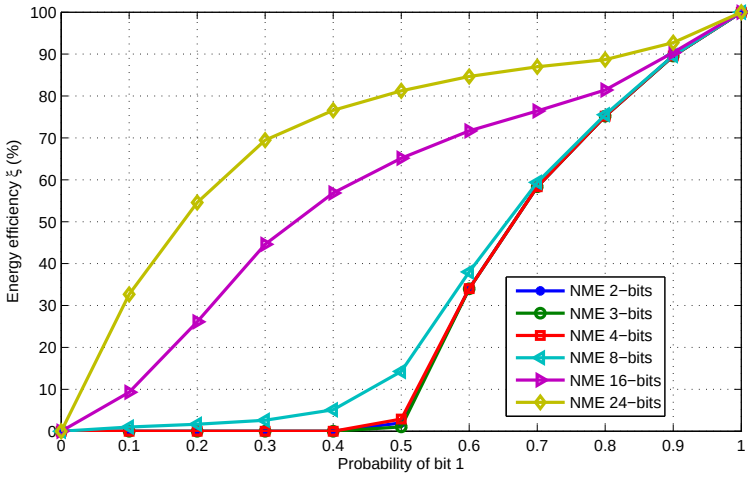



Figure 4.7: Energy efficiency of NME coding for random data with various probabilities of bit 1.

| Coding | Number of 1s in dictionary (bits) | Number of 1s in data (bits) | Number of 1s in total (bits) | Energy efficiency (%) | Dictionary length (byte) | Max dictionary length (byte) |
|---|---|---|---|---|---|---|
| Original | – | – | 132,740 | – | – | – |
| NME 2 bit | 4 | 132,386 | 132,390 | 0.26 | 1 | 1 |
| NME 3 bit | 12 | 132,294 | 132,306 | 0.33 | 3 | 3 |
| NME 4 bit | 32 | 130,010 | 130,042 | 2.03 | 8 | 8 |
| NME 8 bit | 1,024 | 122,955 | 123,979 | 6.60 | 0.25 k | 0.25 k |
| NME 16 bit | 71,808 | 82,463 | 154,271 | −16.22 | 26.7 k | 128 k |
| NME 24 bit | 42,376 | 42,469 | 84,845 | 36.08 | 33 k | 48 M |

Table 4.3: NME performance for `lena.jpg` file (33.2 kB).

*Compressed video*: `news_cif.mp4`. The largest improvement is achieved using NME 8 bit with 2.58% energy efficiency. As a side note, NME 24 bit reduces the number of 1s in coding data, but requires a large dictionary, which generates a negative energy efficiency.

*Uncompressed video*: `bus_qcif.yuv`. NME gains in all cases, and the largest improvement is achieved using NME 16 bit with 54.63% energy efficiency.

*Program file*: `AdobeUpdater.dll`. NME gains in all cases, and the largest improvement is achieved using NME 24 bit with 45.12% energy efficiency.

### Robustness during transmission

The BAC channel error probabilities are set to $p_1 = 0.1$ and $p_2 = 0.004$ (i.e. $p_2$ is 25 times smaller than $p_1$).

*Codeword error rate:* Based on equation (4.10), the codeword error rate for various values of $p_1$, $n$ and $P(1)$ is shown in figure 4.8 (in each case, $p_2 = p_1/25$). For example, when $P(1) = 0.3$ (i.e. 30% of input bits are 1, and 70% are 0) and codeword has 8 bits, the values $p_1 = 0.1$ (probability of receiving 0 when sending 1) and $p_2 = 0.004$ (probability of receiving 1 when sending 0) yield 23% erroneous symbols. As expected, at fixed $1 \to 0$ error probability (a vertical line in the figure), the larger the $n$, the larger the codeword error rate, so the more vulnerable to error during transmission.

*Application to multimedia transmission:* Figure 4.9 presents the robustness during image transmission with various probabilities of error. It can be seen that the bigger the number of bits in NME coding, the greater the efficiency, but the less the robustness/reliability. This is because, as explained in section 4.3.3, the error can increase when using a dictionary like NME does. This can be spotted easily in figure 4.10, which presents the received image using various values for NME coding when transmitting the image through the channel.

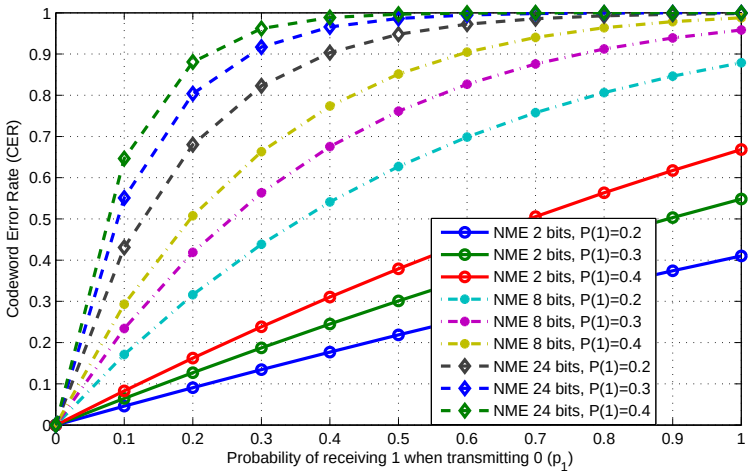



Figure 4.8: Codeword error rate for various parameters.

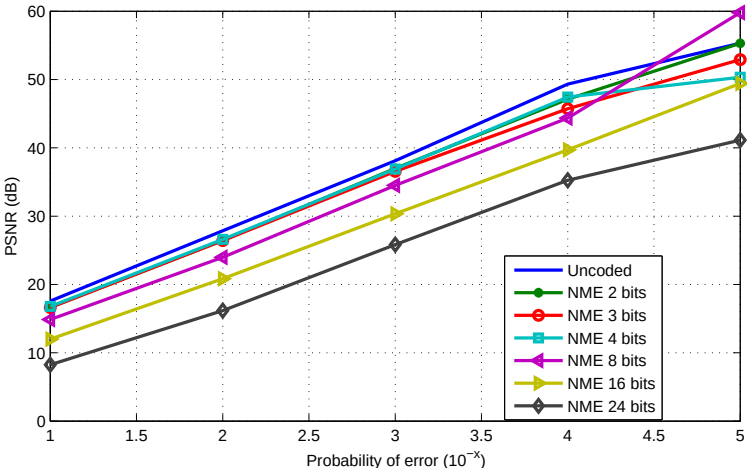



Figure 4.9: PSNR for an image transmission.

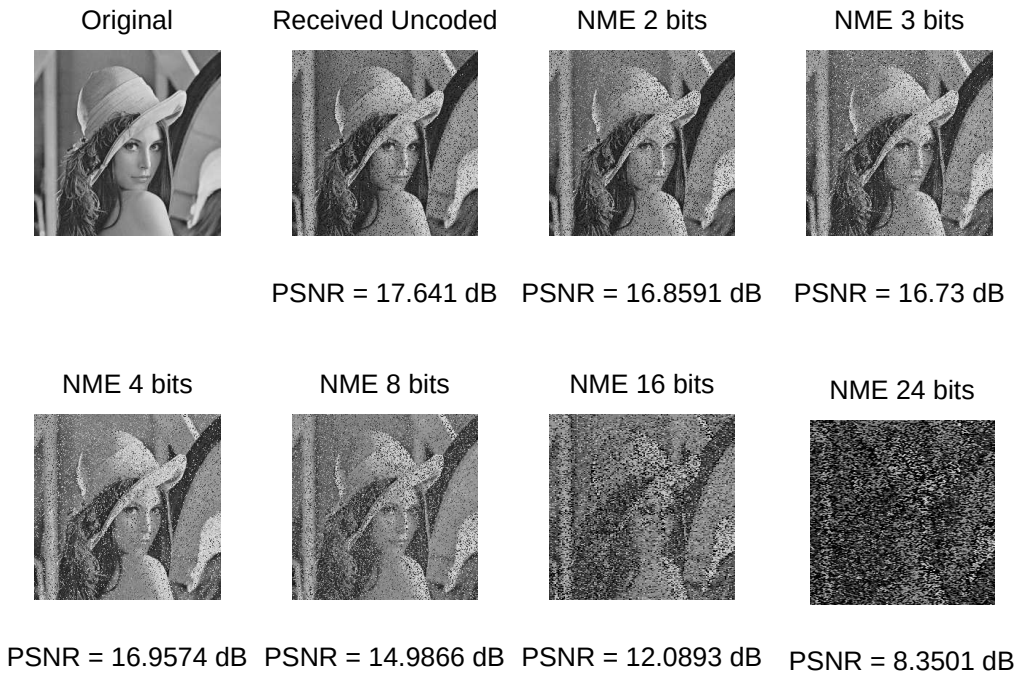



Figure 4.10: Received `lena.bmp` image when transmitted through a BAC channel with $p_1 = 0.1$ and $p_2 = 0.004$.

### 4.3.5 Conclusions

Communication between nanodevices using TS-OOK modulation requires energy to transmit 1 bits and no energy to transmit 0 bits. Since nanosensors transmit much data and have very limited energy, energy requirements of communication is very important. Based on this idea, in this paper we propose a method to reduce the number of 1s in data transmitted, by encoding more often used symbols using fewer 1s. We evaluate it with real files and investigate the code robustness during transmission. Numerical results show that the proposed algorithm saves energy depending on input data distribution, in some of our tests more than 50%, and in theory up to 100%. Results also show that our method is more vulnerable to channel errors, therefore it needs to be combined with error correction code.

## 4.4 Final remarks

Nanotechnology and especially nanowireless communications are fields in their infancy. Not only nanomaterials are at incipient stage, but also and especially the models to understand how they work and how they exchange information. No useful simulator exist for such communications. No nanoantenna radiating at 1 THz and below have yet been built. Assuming that such antennas will be built, my long-term goal is to provide communication protocols for them and make them use in various fields in research, such as video transmission, security and especially in medicine with the aim to make them used by physicians.

# Conclusions and perspectives

This manuscript presented the research work I have done in the last 11 years on network communication. It was divided in four fields, one chapter per field: congestion control, video adaptive streaming, communication in MEMS networks, and communication in nanonetworks. Congestion control chapter presents several various independent ideas to optimise network use. Video adaptive streaming chapter presents mainly three independent ideas: a server-based video adaptation method, a method to reduce oscillations given by video adaptation, and an analysis of video adaptation methods found in the literature. MEMS networks chapter shows the incremental work associated to a funded project, which lead to a functional prototype. Nanonetworks chapter presents some ideas exploiting particularities of nanonetworks.

The aim of almost all the work presented here is optimisation of data transmission. I used various methods to validate the optimisation work. Simulations are the most often used validation method in computer network research, because they allow setup easiness, control, reproducibility and so on. I generally used simulations, but also experiments and numerical results given by numerical computing environments such as Matlab or self-written. Some of the modifications given to simulators have been made available to researchers on my Web page; for example, modifications made to DCCP patch in NS2 simulator, which eventually were integrated into the simulator itself (see section 1.5.6).

Based on broadness of the topic, research papers can be roughly divided in two types: short scope papers presenting a new idea, and broad scope analysis of a field (survey or "philosophic" papers). Most of my work can be counted in the first category, exceptions being [59] and perhaps [54]. Another classification is based on the method used; they can be purely theoretical or based on validation results, or both. Even if some of my work uses formalisation and has a theoretical approach, such as [163], most of my work is technology-driven and validated using tools such as simulators and experimental software.

We live in the age of communication. Network communication is a field a few dozens years old, but is expanding at high speed, sometimes in uncontrollable and unpredictable directions, as shown for example by social networks Web sites. This is fuelled by the day to day activities of millions of people who use videoconferencing, video on demand, to give some examples as of today, and applied research is naturally following this trend. The number of applications and fields using network communication is also increasing and does not seem to stop in the near future, at least; Internet of things include many such initiatives. Digital and network communications have a bright future.

# Bibliography


[1] J. Abrahams. Code and parse trees for lossless source encoding. In *Compression and Complexity of Sequences*, 1997, pages 145–171, Salerno, Italy, June 1997. IEEE.

[2] V. Adzic, H. Kalva, and B. Fuhrt. A survey of multimedia content adaptation for mobile devices. *Multimedia Tools and Applications*, 51(1):379–396, Jan. 2011.

[3] A. G. F. Agarwal, A. Jayaraman, and C. K. Siew. Modified RED gateways under bursty traffic. In *IEEE communications Letters*, volume 8, pages 323–325, 2004.

[4] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Multimedia Systems*, 2, pages 157–168, San Jose, CA, US, Feb. 2011.

[5] I. F. Akyildiz, F. Brunetti, and C. Blázquez. Nanonetworks: A new communication paradigm. *Computer Networks*, 52(12):2260 – 2279, 2008.

[6] I. F. Akyildiz and J. M. Jornet. Electromagnetic wireless nanosensor networks. *Nano Communication Networks*, 1(1):3–19, Mar. 2010.

[7] T. Arsan. Review of bandwidth estimation tools and application to bandwidth adaptive video streaming. In *International Conference on High Capacity Optical Networks and Enabling Technologies (HONET)*, 9, pages 152–156, Istanbul, Turkey, Dec. 2012.

[8] K. Ashton. That 'internet of things' thing. *RFID Journal*, 2009.

[9] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg. Differentiation between short and long TCP flows: predictability of the response time. In *INFOCOM*, volume 2 of 7, pages 762–773, Hong Kong, Mar. 2004. IEEE.

[10] J. Baillieul and P. J. Antsaklis. Control and communication challenges in networked real-time systems. *Proceedings of the IEEE*, 95(1):9–28, Jan. 2007.

[11] H. Balakrishnan and R. Katz. Explicit Loss Notification and Wireless Web Performance. In *IEEE GLOBECOM Global Internet*, Sydney, Australia, Novembre 1998.

[12] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, Dec. 1997.

[13] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *Wireless Networks*, 1(4):469–481, Dec. 1995.

[14] A. Balk, D. Maggiorini, M. Gerla, and M. Y. Sanadidi. Adaptive MPEG-4 video streaming with bandwidth estimation. In *International Workshop on Quality of Service in Multiservice IP Networks*, 2, pages 525–538, London, UK, February 2003. Springer-Verlag.

[15] B. Bamieh, F. Paganini, and M. Dahleh. Distributed control of spatially invariant systems. *Automatic Control, IEEE Transactions on*, 47(7):1091 –1107, jul 2002.

[16] D. Barman and I. Matta. Effectiveness of loss labeling in improving TCP performance in wired/wireless networks. In *ICNP, IEEE International Conference on Network Protocols*, pages 2–11, Washington, DC, USA, 2002. IEEE Computer Society.

[17] S. Biaz and N. H. Vaidya. Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. In *IEEE ASSET Symposium on Application - Specific Systems and Software Engineering and Technology*, pages 10–17, Washington, DC, USA, 1999. IEEE Computer Society.

[18] S. Biaz and X. Wang. Can ECN be used to differentiate congestion losses from wireless losses? Technical Report CSSE04-04, Auburn University, 2004.

[19] S. Biaz and X. Wang. RED for improving TCP over wireless networks. In *ICWN: International Conference on Wireless Networks*, pages 628–636, Las Vegas, USA, June 2004.

[20] D. Biegelsen, A. Berlin, P. Cheung, M. Fromherts, D. Goldberg, W. Jackson, B. Preas, J. Reich, and L. Swartz. Airjet paper mover. In *SPIE Int. Symposium on Micromachining and Microfabrication*, 2000.

[21] N. Bjork and C. Christopoulos. Transcoder architecture for video coding. In *IEEE Trans. Consum. Electron. vol. 44 no. 1*, pages 88–98, 1998.

[22] S. Blake, D. Blak, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated service. RFC 2475, Internet Engineering Task Force, Dec. 1998.

[23] K.-F. Böhringer, V. Bhatt, B. R. Donald, and K. Y. Goldberg. Algorithms for sensorless manipulation using a vibrating surface. *Algorithmica*, 26(3-4):389–429, 2000.

[24] N. Boillot, D. Dhoutaut, and J. Bourgeois. Efficient simulation environment of wireless radio communications in mems modular robots. In *iThings 2013, IEEE Int. Conf. on Internet of Things*, pages 638–645, Beijing, China, Aug. 2013.

[25] N. Boillot, D. Dhoutaut, and J. Bourgeois. Using nano-wireless communications in micro-robots applications. In *International Conference on Nanoscale Computing and Communication*, 1, pages 1–9, Atlanta, Georgia, USA, May 2014. ACM.

[26] F. Borrelli, T. Keviczky, G. J. Balas, G. E. Stewart, K. Fregene, and D. N. Godbole. Hybrid decentralized control of large scale systems. In *HSCC*, volume 3414 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2005.

[27] A. Boukerche, G. Jia, and R. W. N. Pazzi. Performance evaluation of packet loss differentiation algorithms for wireless networks. In *PM2HW2N: Performance monitoring and measurement of heterogeneous wireless and wired networks*, pages 50–52, New York, NY, USA, Octobre 2007. ACM.

[28] J. Bourgeois and S. Goldstein. The internet of [micro]-things. Keynote talk, the 2011 IEEE Int. Conf. on Internet of Things. Dalian, China, Oct. 2011.

[29] J. Bourgeois and S. Goldstein. Distributed intelligent mems: Progresses and perspectives. In L. Kocarev, editor, *ICT Innovations 2011*, volume 150 of *Advances in Intelligent and Soft Computing*, pages 15–25. Springer Berlin / Heidelberg, 2012.

[30] J. Bourgeois and S. C. Goldstein. Distributed intelligent mems: Progresses and perspectives. *IEEE Systems Journal*, 2013.

[31] K. Boutoustous, E. Dedu, and J. Bourgeois. An exhaustive comparison framework for distributed shape differentiation in a MEMS sensor actuator array. In *International Symposium on Parallel and Distributed Computing (ISPDC)*, 7, pages 429–433, Kraków, Poland, July 2008. IEEE. Poster.

[32] K. Boutoustous, E. Dedu, and J. Bourgeois. A framework to calibrate a MEMS sensor network. In *International Conference on Ubiquitous Intelligence and Computing (UIC)*, 6, pages 136–149, Brisbane, Australia, July 2009. Spriger, LNCS 5585.

[33] K. Boutoustous, G. J. Laurent, E. Dedu, L. Matignon, J. Bourgeois, and N. L. Fort-Piat. Distributed control architecture for smart surfaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23, pages 2018–2024, Taipei, Taiwan, Oct. 2010. IEEE.

[34] B. Braden, D. Clark, J. Crowcroft, et al. Recommandations on queue management and congestion avoidance in the internet. IETF standard, Apr. 1998. RFC 2309.

[35] I. S. Burnett, F. Pereira, R. V. de Walle, and R. Koenen, editors. *The MPEG-21 Book*. Wiley, 2006.

[36] L. Campelli, I. F. Akyildiz, L. Fratta, and M. Cesana. A cross-layer solution for ultrawideband based wireless video sensor networks. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–6. IEEE, 2008.

[37] A. Carzaniga, D. Rosenblum, and A. Wolf. Content-based addressing and routing: a general model and its application. Technical report, University of Colorado, Jan. 2000.

[38] S. Cen, P. C. Cosman, and G. M. Voelker. End-to-end differentiation of congestion and wireless losses. *IEEE/ACM Transactions on Networking*, 11(5):703–717, Oct. 2003.

[39] S. Chaumette, R. Laplace, C. Mazel, and R. Mirault. Scual, swarm of communicating uavs at labri: An open uavnet testbed. In *14th International Symposium on Wireless Personal Multimedia Communications, WPMC 2011, Brest, France, October 3-7*, pages 1–5, 2011.

[40] M. Chen and A. Zakhor. Rate control for streaming video over wireless. In *INFOCOM*, volume 2, pages 1181–1190, Hong Kong, Mar. 2004. IEEE Computer and Communication Societies Press.

[41] M.-J. Chen, M.-C. Chu, and C.-W. Pan. Efficient motion-estimation algorithm for reduced frame-rate video transcoder. In *IEEE Transactions on Circuits and Systems for Video Technology Vol. 12, No. 4*, Apr. 2002.

[42] X. Chen and J. Heidemann. Preferential treatment for short flows to reduce web latency. *Computer Networks*, 41(6):779–794, Apr. 2003.

[43] K. Chi, Y. Zhu, X. Jiang, and X. Tian. Optimal coding for transmission energy minimization in wireless nanosensor networks. *Nano Communication Networks*, 4:120–130, 2013.

[44] K. Chi, Y. Zhu, X. Jiang, X. Tian, and V. Leung. Energy-efficient prefix-free codes for wireless nano-sensor networks using ook modulation. *IEEE Transactions on Wireless Communications*, 13(5):2670–2682, May 2014.

[45] C.-F. Chou, M.-W. Hsu, and C.-J. Lin. A trend-loss-density-based differential scheme in wired-cum-wireless networks. In *IWCMC: international conference on Wireless communications and mobile computing*, pages 239–244, Vancouver, British Columbia, Canada, July 2006.

[46] Cisco Systems. Weighted random early detection on the Cisco 12000 series router. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/ios112p/gsr/wred_gs.pdf`, Mar. 2002.

[47] J. M. Cubero, J. Gutiérrez, P. Pérez, et al. Providing 3D video services: The challenge from 2D to 3DTV quality of experience. *Bell Labs Technical Journal*, 16(4):115–134, Mar. 2012.

[48] C. A. da Costa, A. C. Yamin, and C. F. R. Geyer. Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, 7(1):64–73, 2008.

[49] R. D'Andrea and G. Dullerud. Distributed control design for spatially interconnected systems. *Automatic Control, IEEE Transactions on*, 48(9):1478 – 1495, sept. 2003.

[50] L. C. Daronco, V. Roesler, J. V. de Lima, and R. Balbinot. Quality analysis of scalable video coding on unstable transmissions. *Multimedia Tools and Applications*, pages 1–27, 2011. Available online.

[51] L. De Cicco and S. Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering*, USAB'10, pages 447–464, Berlin, 2010. Springer-Verlag.

[52] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Second annual ACM conference on Multimedia systems*, pages 145–156, New York, NY, USA, 2011. ACM.

[53] G. De La Roche, R. Rebeyrortte, K. Jaffrès Runser, and J.-M. Gorce. A New Strategy for Indoor Propagation Fast Computation with MR-FDPF Algorithm. In Acta Press, editor, *Antennas, Radar and Wave Propagation*, Banff Canada, 2005. IASTED.

[54] E. Dedu, G. Bise, and J. Bourgeois. An analysis of congestion controls in centralized control systems. In *International Conference on NETwork Games, COntrol and OPtimization (NetGCooP)*, 6, pages 118–123, Avignon, France, Nov. 2012.

[55] E. Dedu, J. Bourgeois, and K. Boutoustous. Simulation to help calibration of a MEMS sensor network. *International Journal of Pervasive Computing and Communications (IJPCC)*, 6(4):356–372, 2010.

[56] E. Dedu, J. Bourgeois, and M. A. Zainuddin. A first study on video transmission over a nanowireless network. In *International Conference on Nanoscale Computing and Communication*, 1, pages 1–6, Atlanta, Georgia, USA, May 2014. ACM.

[57] E. Dedu, S. Linck, and F. Spies. Removing the MAC retransmission times from the RTT in TCP. In *Euromedia Conference, Workshop on Distributed Multimedia Databases and Multimedia Adaptation*, 11, pages 190–193, Toulouse, France, Apr. 2005. Eurosis.

[58] E. Dedu and E. Lochin. A study on the benefit of TCP packet prioritisation. In *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 17, pages 161–166, Weimar, Germany, Feb. 2009. IEEE.

[59] E. Dedu, W. Ramadan, and J. Bourgeois. A taxonomy of the parameters used by decision methods for adaptive video transmission. *Multimedia Tools and Applications*, ??(??):1–27, ?? 2013.

[60] G. V. der Auwera, P. T. David, M. Reisslein, and L. J. Karam. Traffic and quality characterization of the H.264/AVC scalable video coding extension. *Advances in Multimedia*, 2008:1–27, 2008.

[61] G. V. der Auwera and M. Reisslein. Implications of smoothing on statistical multiplexing of H.264/AVC and SVC video streams. *IEEE Transactions on Broadcasting*, 55(3):541–558, Sept. 2009.

[62] D. Dhoutaut, A. Régis, and F. Spies. Impact of radio propagation models in vehicular ad hoc networks simulations. In *VANET '06: Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, pages 40–49, New York, NY, USA, 2006. ACM.

[63] D. Dhoutaut and F. Spies. Adding geographical interferences into the shadowing pattern model for vehicular ad hoc networks simulations. In *ITST '07. 7th International Conference on Intelligent Transport Systems Communications*, 2007.

[64] Dynamic streaming in Flash Media Server 3.5. Available at `http://www.adobe.com/devnet/flashmediaserver/`, 2010. Adobe Systems Inc.

[65] M. Eberhard, C. Timmerer, H. Hellwagner, and E. Quacchio. An interoperable delivery framework for scalable media resources. *Wireless Communication*, 16:58–63, October 2009.

[66] D. El Baz, V. Boyer, J. Bourgeois, E. Dedu, and K. Boutoustous. Distributed discrete state acquisition and concurrent pattern recognition in a MEMS-based smart surface. In *Workshop on hardware and software implementation and control of distributed MEMS (dMEMS)*, 1, pages 1–8, Besançon, France, June 2010. IEEE.

[67] D. El Baz, V. Boyer, J. Bourgeois, E. Dedu, and K. Boutoustous. Distributed part differentiation in a smart surface. *Mechatronics*, 22(5):522–530, Aug. 2012.

[68] A. Eleftheriadis and B. Anastassiou. Constrained and general dynamic rate shaping of compressed digital video. In *International Conference on Image Processing (ICIP)*, 1995.

[69] C. Erin and H. Asada. Energy optimal codes for wireless communications. In *IEEE Proceedings of the 38th Conference on Decision and Control*, Dec. 1999.

[70] N. Eude, B. Ducourthial, and M. Shawsky. Enhancing ns-2 simulator for high mobility ad hoc networks in car-to-car communication context. In *Proceedings of the 7th IFIP International Conference on Mobile and Wireless Communications Networks (MWCN 2005)*, Morocco, september 2005.

[71] N. Feamster, D. Bansal, and H. Balakrishnan. On the interactions between layered quality adaptation and congestion control for streaming video. In *11th International Packet Video Workshop*, Kyongiu, Korea, April 2001.

[72] M.-W. Feng, S.-L. Wen, K.-C. Tsai, Y.-C. Liu, and H.-R. Lai. Wireless sensor network and sensor fusion technology for ubiquitous smart living space applications (invited paper). In *Proc. Second Int. Symp. Universal Communication ISUC '08*, pages 295–302, 2008.

[73] W. Feng. On the efficacy of quality, frame rate, and buffer management for video streaming across best-effort networks. *Journal of High Speed Networks*, 11:199–214, 2002.

[74] W. Feng and J. Rexford. Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video. *IEEE Transactions on Multimedia*, 1(3):302–312, Sept. 1999.

[75] C. Fischione, K. H. Johansson, A. Sangiovanni-Vincentelli, and B. Z. Ares. Minimum energy coding in CDMA wireless sensor networks. *IEEE Transactions on Wireless Communications*, 8(2):985–994, Feb. 2009.

[76] S. Floyd. Congestion control principles, Sept. 2000. RFC 2914.

[77] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol specification, Sept. 2008. RFC 5348.

[78] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[79] S. Floyd and E. Kohler. Profile for Datagram Congestion Control Protocol (DCCP) congestion control ID 2: TCP-like congestion control. IETF standard, Mar. 2006. RFC 4341.

[80] S. Floyd, E. Kohler, and J. Padhye. Profile for Datagram Congestion Control Protocol (DCCP) congestion control ID 3: TCP-friendly rate control (TFRC). IETF standard, Mar. 2006. RFC 4342.

[81] H. Friis. A note on a simple transmission formula. *Proc. IRE*, 34, 1946.

[82] H. Fujita. Group work of microactuators. In *International Advanced Robot Program Workshop on Micromachine Technologies and Systems*, pages 24–31, Tokyo, Japan, October 1993.

[83] Y. Fukuta, Y.-A. Chapuis, Y. Mita, and H. Fujita. Design, fabrication and control of MEMS-based actuator arrays for air-flow distributed micromanipulation. *IEEE Journal of Micro-Electro-Mechanical Systems*, 15(4):912–926, Aug. 2006.

[84] M. Furini and D. Towsley. Real-time traffic transmission over the Internet. *IEEE Transactions on Multimedia*, 3(1):33–40, Mar. 2001.

[85] J. Gettys and K. Nichols. BufferBloat: What's wrong with the Internet? *Queue*, 9(12):10:10–10:20, Dec. 2011.

[86] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2-4):210–225, 1993.

[87] B. Gorkemli and A. M. Tekalp. Adaptation strategies for streaming SVC video. In *ICIP, The International Conference on Image Processing*, pages 2913–2916, Hong Kong, Septembre 2010.

[88] D. Grossman. New terminology and clarifications for diffserv. Request For Comments 3260, IETF, Apr. 2002.

[89] B. Görkemli and A. M. Takalp. Adaptation strategies for MGS scalable video streaming. *Signal Processing: Image Communication*, 27(6):595–611, July 2012.

[90] E. Gürses, G. B. Akar, and N. Akar. A simple and effective mechanism for stored video streaming with TCP transport and server-side adaptive frame discard. *Computer Networks*, 48:489–501, July 2005.

[91] R. J. Haddad, M. P. McGarry, and P. Seeling. Video bandwidth forecasting. *IEEE Communications Surveys & Tutorials*, PP(99):1–16, Apr. 2013.

[92] A. Haider, H. Sirisena, and K. Pawlikowski. Improved congestion control with hybrid RED. In *10th International Conference on Telecommunications ICT 2003*, volume 2, pages 923–928, 2003.

[93] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification. RFC 3448, Internet Engineering Task Force, Jan. 2003.

[94] T. Haukaas. Rate adaptive video streaming over wireless networks. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2007.

[95] W. P. M. H. Heemels, D. Nesic, A. R. Teel, and N. van de Wouw. Networked and quantized control systems with communication delays. In *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009*, pages 7929–7935, 2009.

[96] J. Hespanha, P. Naghshtabrizi, and Y. Xu. A survey of recent results in networked control systems. *Proceedings of the IEEE*, 95(1):138 –162, jan. 2007.

[97] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21(6):879–894, 2003.

[98] R. Huang, J. Ma, and Q. Jin. A tree-structured intelligence entity pool and its sharing among ubiquitous objects. In *Proc. Int. Conf. Computational Science and Engineering CSE '09*, volume 2, pages 318–325, 2009.

[99] Á. Huszák and S. Imre. Selective retransmission of MPEG video streams over IP networks. In *International Symposium on Communication System Networks and Digital Signal Processing (CSNDSP)*, 5, pages 125–128, Patras, Greece, July 2006.

[100] Information technology – dynamic adaptive streaming over HTTP (DASH) – part 1: Media presentation description and segment formats. ISO standard, Nov. 2011. ISO/IEC 23009-1:2012.

[101] H. Ishida, S. Yanadume, T. Takahashi, I. Ide, Y. Mekada, and H. Murase. Recognition of low-resolution characters by a generative learning method. In *International workshop on camera-based document analysis and recognition (CBDAR)*, 1, pages 45–51, Seoul, Korea, Aug. 2005.

[102] ITU-T. Opinion model for video-telephony applications, Apr. 2007.

[103] R. K. Jain, D.-M. W. Chiu, and W. R. Have. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Research report TR-301, DEC, Littletown, MA, USA, 1984.

[104] Y.-S. Jeong, E.-H. Song, G.-B. Chae, M. Hong, and D.-S. Park. Large-scale middleware for ubiquitous sensor networks. *IEEE Intelligent Systems*, 25(2):48–59, 2010.

[105] S. Jin and A. Bestavros. Gismo: a generator of internet streaming media objects and workloads. *ACM SIGMETRICS Perform. Eval. Rev.*, 29(3):2–10, 2001.

[106] J. M. Jornet. Low-weight error-prevention codes for electromagnetic nanoneworks in the terahertz band. *Nano Communications Networks*, 5(1–2):35–44, mar-jun 2014.

[107] J. M. Jornet and I. F. Akyildiz. The internet of multimedia nano-things. *Nano Communication Networks*, 3(4):242–251, December 2012.

[108] J. M. Jornet and I. F. Akyildiz. Graphene-based plasmonic nano-antenna for terahertz band communication in nanonetworks. *IEEE Journal on Selected Areas in Communications/Supplement—Part 2*, 31(12):685–694, Dec. 2013.

[109] J. M. Jornet and I. F. Akyildiz. Femtosecond-long pulse-based modulation for terahertz band communication in nanonetworks. *IEEE Transactions on Communications*, 62(5):1742–1753, May 2014.

[110] J. M. Jornet and I. F. Akyildiz. Graphene-based plasmonic nano-transceiver for terahertz band communication. In *European Conference on Antennas and Propagation (EuCAP)*, 8, pages 1–5, The Hague, The Netherlands, Apr. 2014. EurAAP.

[111] M. I. Kazantzidis. *Adaptive Multimedia in Wireless IP Networks*. PhD thesis, University of California, Los Angeles, USA, 2002.

[112] F. Kelly. Mathematical modelling of the Internet. In *Mathematics Unlimited — 2001 and Beyond*, pages 685–702, Berlin, Germany, 2001. Springer-Verlag.

[113] J. Kim and J. G. Andrews. An energy efficient source coding and modulation scheme for wireless sensor networks. In *IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, 6, pages 710–714, New York, USA, June 2005. IEEE.

[114] I. Kofler, J. Seidl, C. Timmerer, H. Hellwagner, I. Djama, and T. Ahmed. Using MPEG-21 for cross-layer multimedia content adaptation. *Signal, Image and Video Processing*, 2(4):355–370, 2008.

[115] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). IETF standard, Mar. 2006. RFC 4340.

[116] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340, Mar. 2006.

[117] C. Langbort, R. S. Chandra, and R. D'Andrea. Distributed control design for systems interconnected over an arbitrary graph. *IEEE Trans. Automatic Control*, 2004.

[118] S. Lee and K. Chung. MaVIS: Media-aware video streaming mechanism. In *IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS)*, pages 74–84, Dublin, Ireland, October 2006.

[119] S. Lee and K. Chung. Buffer-driven adaptive video streaming with TCP-friendliness. *Computer Communications*, 31:2621–2630, June 2008.

[120] Z. Lei and N. Georganas. Rate adaptation transcoding for precoded video streams. In *Proceedings of ACM Multimedia*, Juan-les-Pins, France, Dec. 2002.

[121] A. Lie and J. Klaue. Evalvid-RA: trace driven simulation of rate adaptive MPEG-4 VBR video. *Multimedia Systems*, 14(1):33–50, June 2008.

[122] S. Linck, E. Dedu, and F. Spies. Distance-dependent RED policy (DDRED). In *International Conference on Networking (ICN)*, 6, pages 1–6, Sainte-Luce, Martinique, Apr. 2007. IEEE.

[123] S. Linck, E. Mory, J. Bourgeois, E. Dedu, and F. Spies. Video quality estimation of DCCP streaming over wireless networks. In *Euromicro Conference on Parallel, Distributed and Network-based Processing*, 14, pages 405–412, Montbéliard, France, Feb. 2006. IEEE.

[124] S. Linck, E. Mory, J. Bourgeois, E. Dedu, and F. Spies. Adaptive multimedia streaming using a simulation test bed. *Journal of Computational Science*, ?(?):1–22, ? 2014.

[125] C. Liu, I. Bouazizi, and M. Gabbouj. Advanced rate adaption for unicast streaming of scalable video. In *IEEE International Conference on Communications*, pages 1–5, Cape Town, South Africa, May 2010.

[126] C. Liu, I. Bouazizi, and M. Gabbouj. Rate adaptation for adaptive HTTP streaming. In *Second annual ACM conference on Multimedia systems*, pages 169–174, New York, NY, USA, 2011. ACM.

[127] C. Liu, T. Tsao, P. Will, Y. Tai, and W. Liu. A micromachined permalloy magnetic actuator array for micro robotics assembly systems. In *The 8th International Conference on Solid-State Sensors and Actuators*, 1995.

[128] J. E. Luntz and W. Messner. A distributed control system for flexible materials handling. *IEEE Control Systems*, 17(1), February 1997.

[129] J. E. Luntz, W. Messner, and H. Choset. Parcel manipulation and dynamics with a distributed actuator array: The virtual vehicle. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1541–1546, 1997.

[130] J. Ma. Smart u-things–challenging real world complexity. In *IPSJ Symposium Series*, 19, pages 146–150, 2005.

[131] J. Ma, L. Yang, B. Apduhan, R. Huang, L. Barolli, and M. Takizawa. Towards a smart world and ubiquitous intelligence: a walkthrough from smart things to smart hyperspaces and ubickids. *International Journal of Pervasive Computing and Communications*, 1(1):53–68, 2005.

[132] S. Mascolo. Congestion control in high-speed communication networks using the Smith principle. *Automatica*, 35(12):1921 – 1935, 1999.

[133] P. Massioni and M. Verhaegen. Distributed control for identical dynamically coupled systems: A decomposition approach. *Automatic Control, IEEE Transactions on*, 54(1):124 –135, jan. 2009.

[134] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, July 1997.

[135] N.-E. Mattsson. A DCCP module for ns-2. Master's thesis, Luleå University of Technology, Sweden, May 2004.

[136] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *SIGCOMM Computer Communication Review*, 26:117–130, August 1996.

[137] M. Mellia, I. Stoica, and H. Zhang. Tcp-aware packet marking in networks with diffserv support. *Comput. Netw.*, 42(1):81–100, 2003.

[138] G. C. Murilo and M. W. Peter. A general theory for positioning and orienting 2d polygonal or curved parts using intelligent motion surfaces. In *ICRA*, pages 856–862, 1998.

[139] D. Nesic and D. Liberzon. A unified framework for design and analysis of networked and quantized control systems. *IEEE Transactions on Automatic Control*, 54:732–747, 2009.

[140] Network simulator — ns-2. `http://www.isi.edu/nsnam/ns/`.

[141] ns-3. `http://www.nsnam.org`.

[142] D. T. Nguyen and J. Ostermann. Congestion control for scalable video streaming using the scalability extension of H.264/AVC. *IEEE Journal of Selected Topics in Signal Processing*, 1(2):246–253, 2007.

[143] S. Oh, B. Kulapala, A. W. Richa, and M. Reisslein. Continuous-time collaborative prefetching of continuous media. *IEEE Transactions on Broadcasting*, 54(1):36–52, Mar. 2008.

[144] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215 –233, jan. 2007.

[145] R. Pantos and W. May. HTTP live streaming. IETF Draft, Mar. 2011. Apple Inc.

[146] M. K. Park, K.-H. Sihn, and J. H. Jeong. A statistical method of packet loss type discrimination in wired-wireless network. In *CCNC: IEEE Consumer Communications and Networking Conference*, pages 458 – 462, Las Vegas, USA, Jan. 2006.

[147] C. Parsa and J. J. Garcia-Luna-Aceves. Improving TCP congestion control over internets with heterogeneous transmission media. In *ICNP: IEEE International Conference on Network Protocols*, pages 213–221, Toronto, Ontario, Canada, Oct.-Nov. 1999.

[148] G. Piro, L. A. Grieco, G. Boggia, and P. Camarda. Nano-sim: Simulating electromagnetic-based nanonetworks in the network simulator 3. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, pages 203–210, ICST, Brussels, Belgium, 2013.

[149] K. Pister, R. Fearing, and R. Howe. A planar air levitated electrostatic actuator system. In *IEEE Workshop on Micro Electro Mechanical Systems*, pages 61–71, 1990.

[150] J. Postel. User Datagram Protocol. IETF standard, Aug. 1980. RFC 768.

[151] J. Postel. Internet Protocol. RFC 0791, Internet Engineering Task Force, Sept. 1981.

[152] J. Postel. Transmission control protocol. IETF standard, Sept. 1991. RFC 793.

[153] Y. Prakash and S. Gupta. Energy efficient source coding and modulation for wireless applications. In *IEEE WCNC*, 2003.

[154] J. G. Proakis. *Digital Communications*. McGraw-Hill International, 4 edition, 2001.

[155] J. Pu. A survey of the QoS adaptation mechanisms for distributed multimedia systems. Technical report, University of Victoria, Victoria, Canada, Jan. 2000.

[156] I. A. Rai, E. W. Biersack, and G. Urvoy-Keller. Size-based scheduling to improve the performance of short TCP flows. *IEEE Networking*, 19(1):12–17, Jan.-Feb. 2005.

[157] R. Rajesh, M. Lakshmanan, and V. Noor Mohammed. Implementation of networked control systems using TCP/IP. *International Journal of Computer Applications*, 18(2):1–5, Mar. 2011.

[158] W. Ramadan, E. Dedu, and J. Bourgeois. EcnLD, ECN loss differentiation to optimize the performance of transport protocols on wireless networks. In *International Conference on Ultra Modern Telecommunications & Workshops (ICUMT), WMCNT workshop*, 1, pages 1–6, Saint Petersburg, Russia, Oct. 2009. IEEE.

[159] W. Ramadan, E. Dedu, and J. Bourgeois. VAAL, video adaptation at application layer and experiments using DCCP. In *International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 13, pages 1–5, Recife, Brazil, Oct. 2010. Springer.

[160] W. Ramadan, E. Dedu, and J. Bourgeois. Avoiding quality oscillations during adaptive streaming of video. *International Journal of Digital Information and Wireless Communications (IJDIWC)*, 1(1):126–145, Nov. 2011.

[161] W. Ramadan, E. Dedu, and J. Bourgeois. Avoiding zigzag quality switching in real content adaptive video streaming. In *International Conference on Digital Information and Communication Technology and Its Applications (DICTAP)*, 1, pages 421–435, Dijon, France, June 2011. Springer. CCIS 167.

[162] W. Ramadan, E. Dedu, and J. Bourgeois. Oscillation-free video adaptation at application layer on server side and experiments using DCCP. *The Computer Journal*, 57(8):1195–1210, Aug. 2014. Oxford University Press.

[163] W. Ramadan, E. Dedu, D. Dhoutaut, and J. Bourgeois. RELD, RTT ECN Loss Differentiation to optimize the performance of transport protocols on wireless networks. *Telecommunications Systems, special issue on Mobile Computing and Networking Technologies*, 52(4):1797–1817, Apr. 2013.

[164] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, Sept. 2001.

[165] M. Roccetti, V. Ghini, G. Pau, P. Salomoni, and M. E. Bonfigli. Design and experimental evaluation of an adaptive playout delay control mechanism for packetized audio for use over the Internet. *Multimedia Tools and Applications*, 14(1):23–53, May 2001.

[166] S. Roy and B. Shen. Implementation of an algorithm for fast down-scale transcoding of compressed video on the itanium. In *Proceeding of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2002.

[167] T. Sadaf, S. Zareen, F. Iqbal, G. A. Shah, and M. Y. Javed. Link adaptive multimedia encoding in wireless networks: A survey of theory and approaches. In *International Conference on Electronics and Information Engineering*, volume 2, pages V2–5–V2–10, Kyoto, Japan, Aug. 2010. IEEE.

[168] D. Saladino, A. Paganelli, and M. Casoni. A tool for multimedia quality assessment in NS3: QoE Monitor. *Simulation Modelling Practice and Theory*, 32:30–41, Mar. 2013.

[169] J. Sandell, N., P. Varaiya, M. Athans, and M. Safonov. Survey of decentralized control methods for large scale systems. *Automatic Control, IEEE Transactions on*, 23(2):108 – 128, apr 1978.

[170] Sandvine. Global internet phenomena report. `https://www.sandvine.com`, May 1H 2014.

[171] A. Sato, E. Dedu, J. Bourgeois, and R. Huang. Tree-structured knowledge in a distributed intelligent MEMS application. In *Workshop on design, control and software implementation distributed MEMS (dMEMS)*, 2, pages 22–29, Besançon, France, Apr. 2012. IEEE.

[172] T. Schierl and T. Wiegand. H.264/AVC rate adaptation for internet streaming. In *14th International Packet Video Workshop (PV)*, Irvine, CA, USA, December 2004.

[173] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[174] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, Sept. 2007.

[175] C. Semeria. *Supporting Differentiated Service Classes: Queue Scheduling Disciplines*. Juniper Networks, Dec. 2001. White paper.

[176] T. Shanableh and M. Ghanbari. Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats. In *IEEE Transactions on Multimedia, Vol. 2, No. 2*, June 2000.

[177] Y. Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering, Fundamentals, Algorithms, and Standards*. CRC Press, 2007.

[178] B. Skalar. *Digital Communications, Fundamentals and Applications*. Pearson Education Asia, 2001.

[179] I. Stepanov, D. Herrscher, and K. Rothermel. On the impact of radio propagation models on manet simulation results. In *Proceedings of the 7th IFIP International Conference on Mobile and Wireless Communication Networks (MWCN 2005)*, Marrakech, Morocco, september 2005.

[180] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. RFC 2960, Internet Engineering Task Force, Oct. 2000.

[181] T. Stockhammer. Dynamic adaptive streaming over http — standards and design principles. In *ACM conference on multimedia systems*, 2, pages 134–144, Santa Clara, CA, USA, Feb. 2011.

[182] Y.-C. Su, C.-S. Yang, and C.-W. Lee. Optimal FEC assignment for scalable video transmission over burst error channel with loss rate feedback. *Signal Processing: Image Communication*, 18(7):537–547, Aug. 2003.

[183] J. Suh, S. Glander, R. Darling, C. Storment, and G. Kovacs. Combined organic thermal and electrostatic omnidirectional ciliary microactuator array for object positioning and inspection. In *Solid State Sensor and Actuator Workshop*, 1996.

[184] S. Tabbone, L. Wendling, and J.-P. Salmon. A new shape descriptor defined on the Radon transform. *Computer Vision and Image Understanding*, 102(1):42–51, Apr. 2006.

[185] Y. Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda. Achieving moderate fairness for UDP flows by path-status classification. In *LCN: IEEE Local Computer Networks*, pages 252–264, Washington, DC, USA, Aug. 2000.

[186] B. Vandalore, W. chi Feng, R. Jain, and S. Fahmy. A survey of application layer techniques for adaptive streaming of multimedia. *Real-Time Imaging*, 7(3):221–235, June 2001.

[187] P. Voulgaris. A convex characterization of classes of problems in control with specific interaction and communication structures. In *American Control Conference, 2001. Proceedings of the 2001*, volume 4, pages 3128 –3133, 2001.

[188] N. Wakamiya, M. Murata, and H. Miyahara. TCP-friendly video transfer. In *Internet Quality and Performance and Control of Network Systems*, volume 4211, pages 25–35. SPIE, 2001.

[189] P. Wang, J. M. Jornet, M. A. Malik, N. Akkari, and I. F. Akyildiz. Energy and spectrum-aware mac protocol for perpetual wireless nanosensor networks in the terahertz band. *Ad Hoc Networks*, 11(8):2541–2555, 2013.

[190] M. Weiser. The computer for the 21st century. *IEEE Pervasive Computing*, 99(1):19–25, 2002.

[191] M. Wien, R. Cazoulat, A. Graffunder, A. Hutter, and P. Amon. Real-Time system for adaptive video streaming based on SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1227–1237, 2007.

[192] M. Wien, H. Schwarz, and T. Oelbaum. Performance analysis of SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1194–1203, Sept. 2007.

[193] J. Wu and T. Chen. Design of networked control systems with packet dropouts. *Automatic Control, IEEE Transactions on*, 52(7):1314 –1319, july 2007.

[194] B. Xie and W. Zeng. Rate-distortion optimized dynamic bitstream switching for scalable video streaming. In *IEEE ICME, International Conference on Multimedia and Expo*, volume 2, pages 1327–1330, Taipei, Taiwan, 2004.

[195] J. Xin and C.-W. L. M.-T. Sun. Digital video transcoding. In *Proceedings of the IEEE, Vol. 93, No. 1*, Jan. 2005.

[196] T. Yang. Networked control system: a brief survey. *Control Theory and Applications, IEE Proceedings* -, 153(4):403 – 412, july 2006.

[197] D. Ye, J. C. Barker, Z. Xiong, and W. Zhu. Wavelet-based VBR video traffic smoothing. *IEEE Transactions on Multimedia*, 6(4):611–623, Aug. 2004.

[198] G. Ye, T. Saadawi, and M. J. Lee. Improving stream control transmission protocol performance over lossy links. *Selected Areas in Communications, IEEE Journal on*, 22(4):727–736, 2004.

[199] J. Yin, T. ElBatt, G. Yeung, B. Ryu, S. Habermas, H. Krishnan, and T. Talty. Performance evaluation of safety applications over dsrc vehicular ad hoc networks. In *in Proceedings of VANET*, pages 1–9, 2004.

[200] J. Youn, M.-T. Sun, and C.-W. Lin. Motion vector refinement for high-performance transcoding. In *IEEE Transactions on Multimedia Vol. 1*, Mar. 1999.

[201] J. Youn, M.-T. Sun, and J. Xin. Video transcoder architectures for bit rate scaling of H.263 bit streams. In *ACM Multimedia*, pages 243–250, Nov. 1999.

[202] T. Young. High quality video conferencing. Honours project, University of Waikato, Hamilton, New Zealand, Dec. 2003.

[203] M. A. Zainuddin, E. Dedu, and J. Bourgeois. NME, Nanonetwork Minimum Energy coding. In *IEEE International Conference on Ubiquitous Intelligence and Computing (UIC)*, 11, pages 1–8, Bali, Indonesia, Dec. 2014. IEEE. **Under review**.

[204] A. Zambelli. IIS Smooth Streaming technical overview, Mar. 2009. Microsoft Corporation.

[205] B. Zeng and M. Atiquzzaman. DSRED: an active queue management scheme for next generation networks. In *25th Annual IEEE Conference on Local Computer Networks LCN 2000*, pages 242–251, 2000.

[206] D. Zhang, H.; Ferrari. Rate-controlled static-priority queueing. In *In Proceedings of IEEE Infocom 1993*, Mar. 1993.

[207] Z. Zheng and R. Kinicki. Adaptive explicit congestion notification. In *Seventh International Symposium on Computers and Communications (ISCC'02)*, pages 855–860, Italy, July 2002.

[208] Q.-F. Zhu, L. Kerofsky, and M. B. Garrison. Low-delay, low-complexity rate reduction and continuous presence for multipoint videoconferencing. In *IEEE Transactions on circuits and systems for video technology Vol. 9 No 4*, June 1999.