Some Performance and Design Aspects of Overlay Networks

Ph.D. Thesis

by

Csaba Király

Advisors:

Dr. Tien Van Do, Budapest University of Technology and Economics
Dr. Renato Lo Cigno, University of Trento

Budapest, Hungary
February 13, 2012

© Csaba Király 2012
Abstract

In recent years, services built using application level overlay network technologies have seen a tremendous increase both in their number and in the sheer amount of bytes transferred. The general objective of this thesis is to identify key performance issues and bottlenecks of these overlay networks, to analyze their impact on state of the art systems, and to provide new techniques to improve performance. The thesis is divided into two main parts.

In the first part, performance characteristics of Anonymous Routing Overlays are studied. Anonymous Routing Overlays provide privacy enhancing services, protecting the identity of communicating parties both from each other and from eavesdroppers. The most well known example of such systems is Tor, used by hundreds of thousands of people all over the world. TCP is used in Tor (and several other system designs) as the overlay link technology of choice. First, a novel method for the analytical evaluation of distributions (and quantiles) of the completion times of short-lived TCP connections is presented. Based on the observation of performance issues rooted in the use of TCP as overlay tunnel technology, we introduce IPpriv, a novel anonymous overlay network design to overcome these problems. IPpriv is a datagram-based design that entirely builds on standard IPsec features. We confront IPpriv with Tor through controlled experiments, demonstrating that the new design provides significant improvements in transmission delays.

In the second part of the thesis, we study live P2P Streaming systems. We concentrate our attention on the scheduling problem of mesh type chunk-based systems. The new DLc/ELp scheduling algorithm is introduced, and we formally prove that this algorithm achieves strict optimality under idealized conditions. According to the best of our knowledge, this is the first distributed algorithm to achieve such bound. In following chapters, assumptions of the idealized model are relaxed. First, bandwidth heterogeneity is studied, and a new bandwidth-aware scheduler is introduced and shown to outperform other algorithms from literature. Then, the real structure of the media stream is considered, and Quality of Experience metrics are derived. We close the thesis with the introduction and detailed evaluation of media-aware scheduling algorithms.
## Contents

Abstract ................................................................. iii

Contents ............................................................... v

List of Figures ......................................................... ix

Acknowledgements ..................................................... xiii

List of Abbreviations ................................................. xv

1 Introduction ......................................................... 1
   1.1 Anonymous Routing Overlays ................................. 2
   1.2 Peer-to-Peer Streaming Overlays .............................. 3
   1.3 Thesis Structure ............................................... 4

I Anonymous Routing Overlays ................................. 7

2 Performance of TCP-based Overlay Tunnels ................... 9
   2.1 Introduction .................................................... 9
   2.2 Previous Work .................................................. 12
   2.3 Open Multiclass Queuing Network Model of TCP .......... 13
      2.3.1 Customer’s Behavior in the OMQN Model .............. 14
   2.4 Computation of the Completion Time Distribution ......... 16
   2.5 Validation and Results ........................................ 23
      2.5.1 Model Validation ....................................... 24
      2.5.2 Further Results ....................................... 30
   2.6 Summary ....................................................... 31

3 IPsec-Based Anonymous Networking ............................ 33
   3.1 Background and Privacy Protection ......................... 33
   3.2 Principles of Anonymous Networking ....................... 34
II Peer-to-Peer Streaming Overlays

4 On the Optimality of Scheduling Algorithms

4.1 Introduction

4.2 Problem Statement

4.2.1 System Description

4.2.2 Formal Notation and Definitions

4.2.3 On Quantiles and Other Statistics

4.2.4 Scheduling Peers and Chunks

4.3 Related Work and Contributions

4.4 Optimal Peer Scheduling

4.4.1 Analysis of ELp

4.5 Optimal Chunk Scheduling

4.5.1 Analysis of DLc with Full Meshes

4.6 Neighborhood Restriction and Selected Results

4.6.1 Simulating P2P Streaming and Measuring Performance

4.6.2 Restricting the Overlay

4.6.3 Limiting the Chunk Buffer Size

4.7 Summary

5 Bandwidth-Aware Scheduling Algorithms

5.1 Introduction

5.2 A Bandwidth-Aware ELp Algorithm

5.3 Network and Bandwidth Models

5.4 Comparison of Algorithms

5.5 Sensitivity Analysis with respect to $\beta$ and $\delta$
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6 Summary</td>
</tr>
<tr>
<td><strong>6 Media-Aware Scheduling</strong></td>
</tr>
<tr>
<td>6.1 Introduction</td>
</tr>
<tr>
<td>6.2 Video Encoding and Chunkisation</td>
</tr>
<tr>
<td>6.3 Quantitative Quality Evaluation</td>
</tr>
<tr>
<td>6.4 Experimental Results</td>
</tr>
<tr>
<td>6.5 Summary</td>
</tr>
<tr>
<td><strong>7 Deadline-Based Sub-Stream Scheduling</strong></td>
</tr>
<tr>
<td>7.1 Extending DLc to Sub-Stream Scheduling</td>
</tr>
<tr>
<td>7.2 Evaluation Scenarios</td>
</tr>
<tr>
<td>7.3 Algorithm Validation</td>
</tr>
<tr>
<td>7.3.1 Sub-Stream Differentiation</td>
</tr>
<tr>
<td>7.3.2 Robustness to Chunk Loss</td>
</tr>
<tr>
<td>7.4 Performance Evaluation</td>
</tr>
<tr>
<td>7.4.1 Received Video Quality</td>
</tr>
<tr>
<td>7.5 Summary</td>
</tr>
<tr>
<td><strong>8 Conclusions</strong></td>
</tr>
<tr>
<td>8.1 Summary of new results</td>
</tr>
<tr>
<td>8.1.1 Performance of Anonymous Routing Overlays</td>
</tr>
<tr>
<td>8.1.2 Performance of Peer-to-Peer Streaming Overlays</td>
</tr>
<tr>
<td>8.2 Applicability of results</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
</tr>
<tr>
<td><strong>Related Publications</strong></td>
</tr>
</tbody>
</table>
List of Figures

2.1 Samples of connection evolution within a simplified TCP OMQN model ... 15
2.2 The OMQN model of TCP NewReno ........................................ 17
2.3 Sketch of the recursive procedure to derive the probability mass function of $Y(s)$ 21
2.4 Impact of $R$ on the distribution of the TCP connection completion time ... 21
2.5 Pseudocode of the iterative algorithm for the computation of the completion time distribution ........................................ 23
2.6 Impact of $N$ on the computation time of the FPA and of the TCP connection completion time distribution ........................................ 24
2.7 Single bottlenecks scenario with homogeneous connections. ............... 25
2.8 CDF and CCDF of the completion time; Single bottleneck, homogeneous connections with $N_s = 100$, $\rho_n = 0.9$; Analysis ($\text{RTT}=134\text{ ms}$, $P_L = 1.64\%$) and simulation ($\text{RTT}=132\text{ ms}$, $P_L = 1.61\%$). ......................... 26
2.9 Analysis and simulation. Probability density function. Single bottleneck, homogeneous connections with $N_s = 100$, $\rho_n = 0.9$. ........................................ 27
2.10 CCDF, single bottleneck, connections with $N_s = 10$, $\rho_n = 0.95$. Analysis ($\text{RTT}=117\text{ ms}$, $P_L = 0.6\%$) and simulation ($\text{RTT}=109\text{ ms}$, $P_L = 0.5\%$). ........ 27
2.11 Two bottlenecks scenario, non-homogeneous connections. Setup (left) and simulation results (right). Flows from router 2 to router 1 ($\text{RTT}=47\text{ ms}$, $P_L = 0.7\%$) and from router 3 to router 1 ($\text{RTT}=95\text{ ms}$, $P_L = 2.7\%$). ......................... 28
2.12 Analysis and simulation. CCDF. Two bottlenecks scenario, non-homogeneous connections. Connection length $N_s = 100$ (left) and $N_s = 10$ (right). .......... 29
2.13 CDF and CCDF. Single bottleneck scenario, mixed UDP and TCP traffic. Connection length $N_s = 10$. Analysis and simulation ($\text{RTT}=175\text{ ms}$, $P_L = 29\%$). 30
2.14 Analytical results: mean of the completion time as a function of the connection length and of the segment loss probability. .......................... 31
2.15 Analytical results: 0.95 (left) and 0.99 (right) completion time quantiles as a function of the connection length $N_s$ and of the segment loss probability $P_L$. 32
<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Protecting anonymity in face of the destination and of an external attacker with a favorable observation point where several links and routers can be controlled.</td>
</tr>
<tr>
<td>3.2</td>
<td>Anonymous networking realized with Onion Routing</td>
</tr>
<tr>
<td>3.3</td>
<td>Structure of onion encrypted packet leaving the source and subsequent packet transformations (ESP trailer omitted for simplicity); the shaded part of the packets is encrypted and thus not 'visible' to ORs.</td>
</tr>
<tr>
<td>3.4</td>
<td>Testbed configuration and experiment setup</td>
</tr>
<tr>
<td>3.5</td>
<td>Traffic received to complete the web page download</td>
</tr>
<tr>
<td>3.6</td>
<td>CNN web page download time</td>
</tr>
<tr>
<td>4.1</td>
<td>General and fully connected mesh topologies</td>
</tr>
<tr>
<td>4.2</td>
<td>Full mesh overlay; maximum diffusion delay as a function of N; 500 chunks</td>
</tr>
<tr>
<td>4.3</td>
<td>Worst case chunk diffusion delay of algorithms, with 1000 peers, as a function of: (left) neighborhood size, with 2000 chunks; (right) number of chunks, with $N_N = 11$.</td>
</tr>
<tr>
<td>4.4</td>
<td>Chunk loss and F as a function of the neighborhood size ($N = 10000, D = 32$)</td>
</tr>
<tr>
<td>5.1</td>
<td>The ELp algorithm.</td>
</tr>
<tr>
<td>5.2</td>
<td>3-class scenario with full mesh: diffusion delay as a function of the heterogeneity factor $h$; Upper plot: $F_{90}$, Lower plot: $F$; $\overline{B} = 1$, $N = 600$ peers in full mesh, $M_c = 1200$ chunks.</td>
</tr>
<tr>
<td>5.3</td>
<td>$F_{90}$ as a function of heterogeneity for neighbourhood size 20 and playout delay 50; Upper plot: 3-class, Lower plot: Uniform; $\overline{B} = 1$, $N = 600$ peers, $M_c = 2000$ chunks.</td>
</tr>
<tr>
<td>5.4</td>
<td>$F_{90}$ as a function of $\overline{B}$ for neighbourhood size 20 and playout delay 50; Upper plot: 3-class with $h = 0.5$, Lower plot: Uniform with $\Delta_B = 0.8$; $N = 1000$ peers, $M_c = 2000$ chunks.</td>
</tr>
<tr>
<td>5.5</td>
<td>Free Riders scenario, $\overline{B} = 1$, neighbourhood size 100 and playout delay 50: $F_{90}$ versus the fraction of the free riders. $\overline{B} = 1$, $N = 1000$ peers, $M_c = 2000$ chunks.</td>
</tr>
<tr>
<td>5.6</td>
<td>BA$\beta$ELp sensitivity to $\beta$: 3-class scenario with $h = 0.5$ and $\overline{B} = 1$, $N = 1000$, $M_c = 2000$, neighborhood size 20 and playout delay 50$T$, different chunk schedulers</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>5.7</td>
<td>DLδc sensitivity to δ for different heterogeneity factors h; $\overline{\beta} = 1$, $N = 1000$, $M_c = 2000$, neighborhood size 20 and playout delay $50T_s$, BA3ELp peer scheduler</td>
</tr>
<tr>
<td>5.8</td>
<td>BAβELp sensitivity to β, for different heterogeneity h; $\overline{\beta} = 1$, $N = 1000$, $M_c = 2000$, neighborhood size 20 and playout delay $50T_s$</td>
</tr>
<tr>
<td>6.1</td>
<td>The quality evaluation tool</td>
</tr>
<tr>
<td>6.2</td>
<td>Chunk loss rate as a function of video bitrate and playout delay</td>
</tr>
<tr>
<td>6.3</td>
<td>PSNR with media-aware and unaware chunkizers, as a function of video coding rate, at various target playout delays</td>
</tr>
<tr>
<td>6.4</td>
<td>SSIM with media-aware and unaware chunkizers, as a function of video coding rate, at various target playout delays</td>
</tr>
<tr>
<td>6.5</td>
<td>Chunk loss rate and PSNR with different schedulers and GOP sized chunks, as a function of playout delay</td>
</tr>
<tr>
<td>6.6</td>
<td>SSIM achieved using different codecs with GOP chunkisation, LUc/RUp scheduling and $T_s = 14$</td>
</tr>
<tr>
<td>7.1</td>
<td>sub-stream differentiation using DLc scheduler’s deadline increment and PLUc</td>
</tr>
<tr>
<td>7.2</td>
<td>loss ratio and maximum delay of chunks belonging to different sub-streams, using 4 sub-stream encoding</td>
</tr>
<tr>
<td>7.3</td>
<td>Chunk size variation over the first 100 chunks: 31.7% of the bytes are in I chunks, 46.9% are in P chunks and 21.4% are in B chunks</td>
</tr>
<tr>
<td>7.4</td>
<td>Chunk loss for different sub-streams, as a function of level of prioritisation</td>
</tr>
<tr>
<td>7.5</td>
<td>Chunk loss for various peer classes and chunk priority classes</td>
</tr>
<tr>
<td>7.6</td>
<td>per frame PSNR values with sub-streams (three separate chunks for the I,P and B frames of a GoP) and without sub-streams (1 chunk = 1 GoP)</td>
</tr>
<tr>
<td>7.7</td>
<td>PSNR for all peers, and at different peer classes as a function of prioritization; average for all peers; high-, mid- and low-bandwidth peers</td>
</tr>
<tr>
<td>7.8</td>
<td>PSNR increase changing from PLUc to DL∗c in different peer classes</td>
</tr>
<tr>
<td>7.9</td>
<td>PSNR with different schedulers, at different peer classes, as a function of video encoding rate</td>
</tr>
</tbody>
</table>
Acknowledgements

Without going into details and lengthy descriptions of all their merits, I would like to express my thanks to Tien Van Do, Renato Lo Cigno, László Jereb, Luca Abeni, Marco Ajmone Marsan, Marco Mellia, Emilio Leonardi, Michela Meo, Stefano Traverso, Zsolt Pándi, Roberto Cascella, Krisztina Mahóné, Roberta Guidolin, and of course to Mom, Dad, Zita, Tomi, Győző, Jutka, and Berni.
List of Abbreviations
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>ADSL</td>
<td>Asymmetric Digital Subscriber Line</td>
</tr>
<tr>
<td>AG</td>
<td>Anonymity Gateway</td>
</tr>
<tr>
<td>AN</td>
<td>Anonymous Network</td>
</tr>
<tr>
<td>BAELp</td>
<td>Bandwidth-Aware Earliest-Latest peer scheduler</td>
</tr>
<tr>
<td>CCDF</td>
<td>Complement of the Cumulative Distribution Function</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative Distribution Function</td>
</tr>
<tr>
<td>DLc</td>
<td>Deadline-based chunk scheduler</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>EDF</td>
<td>Earliest Deadline First</td>
</tr>
<tr>
<td>ELp</td>
<td>Earliest-Latest peer scheduler</td>
</tr>
<tr>
<td>EN</td>
<td>Exit Node</td>
</tr>
<tr>
<td>ER</td>
<td>Erdős-Rényi (ER) random graph</td>
</tr>
<tr>
<td>FPA</td>
<td>Fixed Point Approximation</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>IPsec</td>
<td>Internet Protocol Security</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>LUc</td>
<td>Latest Useful chunk scheduler</td>
</tr>
<tr>
<td>MDp</td>
<td>Most Deprived peer scheduler</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>OMQN</td>
<td>Open Multiclass Queuing Network</td>
</tr>
<tr>
<td>OR</td>
<td>Overlay Router</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal to Noise Ratio</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RTT</td>
<td>Round-Trip Time</td>
</tr>
<tr>
<td>RUc</td>
<td>Random Useful chunk scheduler</td>
</tr>
<tr>
<td>RUp</td>
<td>Random Useful peer scheduler</td>
</tr>
<tr>
<td>SA</td>
<td>IPsec Security Association</td>
</tr>
<tr>
<td>SP</td>
<td>IPsec Security Policy</td>
</tr>
<tr>
<td>SSIM</td>
<td>Structural Similarity Index</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>VoD</td>
<td>Video on Demand</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In recent years, services built using overlay network technologies have seen a tremendous increase both in their number and in the sheer amount of bytes transferred. Proliferation of broadband access, stronger CPUs and advances in high-level programming languages (such as simplified networking APIs and libraries) have led to system designs that handle the network with a high level of abstraction, masquerading network connections as simple pipes, making the application almost independent, but also unaware of the underlying networking technologies. This trend has led to systems where several networking functionalities are implemented in application layer and the interaction between these high level algorithms and the network itself is a decisive factor in system performance.

An overlay network is conceptually a network constructed above another networking layer. Overlays are typically implemented in order to provide advanced routing, querying, data manipulation or security features that are not present in the underlying networking layer. Application level overlay networks can be formed for many reasons, the most typical being lack of control over the resources and capabilities of the underlying networking layer.

Nodes of an application level overlay network are all IP end-points. Thus, contrary to the IP level, connections between these nodes are virtual and can be reorganized almost at will. Several encapsulation techniques can be used to implement these connections (overlay link functionality), from IPsec tunnels through custom UDP based protocols to TCP or even HTTP/TCP tunnels.

In an idealized Internet, all of the overlay nodes would have public IP addresses and they would be able to connect to each other should need arise. In practice, these connections are limited due to connectivity constraints on the Internet introduced by NAT (Network Address Translation) and firewalls. Nevertheless, the flexibility of the system far outreaches that of routing at the IP level, as far as features are concerned.

On the performance side, the picture is much less promising for three main reasons:

• Overlay routers are typically running on the edges of the Internet, on home PCs or, at best, in data-centers. Whichever the case, their position is far less central than that of core IP routers, resulting in longer links between routers and duplicated
traffic at the IP layer.

- Overlay routers are usually implemented in application space (contrary to the kernel-space implementation of IP routing or to dedicated routing hardware). This necessarily impedes routing performance.

- Overlay links are implemented above the IP layer, adding overhead to the traffic.

Two prominent examples of the power of such overlay networks are Anonymous Routing Overlays and Peer-to-Peer Streaming Overlays. In both cases, the use of application layer overlay techniques allowed the quick development and introduction of large scale distributed systems, boosting vast areas of research, experimentation, as well as real services.

1.1 Anonymous Routing Overlays

Anonymous Routing Overlays provide privacy enhancing services, protecting the identity of communicating parties both from each other and from eavesdroppers or other malicious attackers. When we contrast this to the amount of information revealed in a single IP packet on the Internet, it becomes evident how fundamental this service is in certain use cases, most notably to enable freedom of speech over the Internet. On the Internet, when node A sends an IP packet to node B, the packet will contain both A’s and B’s IP address in clear form all along the path. This means that A’s identity is revealed to B, therefore services that would require anonymity such as anonymous voting or anonymous payment are difficult to implement. Even if anonymity is granted at higher levels, the IP layer reveals real identities, potentially breaking all the protection provided in higher layers. Sending A’s and B’s address also means that B’s identity is revealed to A’s ISP, and indirectly to A’s government, hindering privacy and allowing the detailed profiling of individuals. Finally, anyone eavesdropping on any single point of the path becomes aware of the communication between the two parties.

In order to implement privacy enhancing services, the routing path should be manipulated and packets should be modified in each routing step. None of these functionalities are controllable by the end-user of the Internet directly at the level of IP routing, thus another layer is introduced with application level routing functionality to implement these features.

1 in the case of a single-homed machine used as an overlay router, inward and outward copies of routed data necessarily cross the same IP (and lower) level link
The best known example of such overlay systems is Tor \[21\], used by hundreds of thousands of people all over the world. Although Tor is already deployed as a service, it is known to suffer from serious performance bottlenecks \[20\]: available bandwidth is relatively low and end-to-end delays can be an order of magnitude higher compared to normal IP, effectively limiting Tor’s applicability to only a few applications.

### 1.2 Peer-to-Peer Streaming Overlays

P2P streaming and in particular P2P distribution of live TV channels is becoming not only a hot research topic, but it is also available as systems and services, see \[28–30, 37, 49\]. The term P2P streaming is used both for the distribution of live content and for large scale VoD (Video-on-Demand) systems. The two, however, present largely different research challenges.

Live P2P streaming overlays can potentially reach sizes of hundreds of thousands of users watching the same channel at the same time. Fundamental to support live streaming is to guarantee the distribution of the continuously generated stream of information to all peers with a low delay. This delay is strictly related to the overlay characteristics and to the scheduling algorithms that distribute chunks (pieces of the stream) to peers. For the system to be efficient, peers’ upload resources should be utilized wisely, while each peer’s download rate is naturally limited by the rate of the video stream.

In the case of VoD, the focus is on distributed storage of video files, on indexing and retrieval of meta information, and on the fast distribution of these files from those already caching the content (seeders) to those interested in it (leechers). Delivery order of pieces is important in order to allow early starting of video playback, but distribution delay is not that important. Peers with more download bandwidth can download the file faster than playback would require, and peers with less resources can cache large part of the content before starting playback. Swarms (set of peers interested in the same content) are usually much smaller than in the case of live streaming.

In a Peer-to-Peer streaming overlay, the required functionality is similar to that of IP level multicast. Ideally, IP multicast could resolve the problem of delivering each packet of a multimedia stream to every listener, in fact it is being used in many ISP’s IP-TV system for such purpose. However, lack of control from the end-user and IP level multicast’s inability to cross administrative boundaries impedes its application on the Internet scale. Thus, application layer solutions are used to provide multicast-like functionality. By moving functionality to the upper layer, other features can be embedded as well, such as data-driven routing and new models of assigning priorities to
information pieces.

In this thesis we only deal with the problem of live P2P streaming, tackling the problem of designing and implementing efficient chunk and peer schedulers.

1.3 Thesis Structure

The general objective of this thesis is to identify key performance issues and bottlenecks of overlay networks, analyze their impact on state of the art systems, and to provide new techniques to improve performance. The thesis is divided into two main parts.

The goal of the first part is to analyze performance characteristics of Anonymous Routing Overlays, and to overcome their most important performance problem, i.e. high end-to-end delays.

In Chapter 2 we provide deeper insight into the transport mechanisms hidden behind some of these abstract overlay technologies by showing an analysis of TCP dynamics in the case of short-lived connections, deriving detailed performance characteristics (such as completion time distribution) through an open multiclass queuing network model of TCP.

Performance differences between TCP based and datagram based overlay networks are then studied in Chapter 3 through the example of Anonymous Routing Overlays. We propose a novel system design for anonymous routing, called IPpriv, an overlay network similar in its scope to Tor, but different in its design choices and performance characteristics. The peculiarity of IPpriv is not just its datagram based design approach, but that it entirely relies on IPsec, therefore providing improved performance due to kernel level or router based operation.

In the second part of the thesis another widely used example of overlay networks is considered: peer-to-peer (P2P) networks. More specifically, our goal is to analyze and improve the performance of live P2P Streaming Overlays.

In P2P streaming, scheduling is the decision of what to send and whom to send it to. Chapter 4 formalizes the scheduling problem for live P2P streaming and introduces some performance bounds. Then, it presents two novel scheduling algorithms: one for the selection of peers and the other for the selection of chunks. We prove that the combination of these two yields optimal performance in idealized conditions.

Chapter 5 extends the peer scheduler of Chapter 4 adding peer bandwidth awareness to it. This extension allows us to improve performance in networks with largely heterogeneous peer upload bandwidth distribution, the realistic case on today’s Internet with a large variety of access technologies.
Chapters 6 and 7 tackle another problem often neglected in the design of scheduling algorithms: the structure and uneven bitrate distribution of media streams. It extends results of Chapter 4 to the case of uneven chunk sizes and differentiated chunk priorities. Comparison of different algorithms at this level of detail is impossible without quantifying the quality of the received media, therefore it also extends the evaluation framework by adding evaluation based on real video traces and deriving video quality measures.

All software used in the thesis is available to the research community as open-source code, both for further research and for the reproducibility of the results presented herein. A real world implementation of the P2P streaming framework is also available and has been used to broadcast various TV channels and events live over the Internet. All related software has been developed in the framework of the DISCREET FP6 and NAPA-WINE FP7 European research projects. The author of this thesis is co-author of the simulators and of the video quality evaluation framework and of the PeerStreamer streaming application.
Chapter 1. Introduction
Part I

Anonymous Routing Overlays
Chapter 2

Performance of TCP-based Overlay Tunnels

2.1 Introduction

Several overlay designs \[21, 55\] use TCP to connect overlay nodes. There are a number of reasons that support such a choice: these designs can leverage congestion and flow control features of TCP; they can also rely on ensured delivery that simplifies encryption schemes; last, but not least, the use of TCP provides a simple programming interface. Some designs even use HTTP over TCP \[18\] to simplify firewall traversal and thus improve connectivity in the overlay graph.

While TCP is a reasonable choice for all the above reasons, it is important to highlight that performance of TCP has been mostly studied in steady-state or in asymptotical conditions. Less is known about TCP performance for short interactions (also called short-lived connections), typical in overlays that are frequently reorganized.

In this chapter, a new technique for the analytical evaluation of distributions (and quantiles) of the completion time of short-lived TCP connections is presented and discussed. The proposed technique derives from known open multiclass queuing network (OMQN) models of the TCP protocol and computes a discrete approximation, with arbitrary accuracy, of the distribution of sojourn times of customers in the OMQN, which corresponds to the distribution of completion times of the modeled TCP connections. The proposed technique is computationally efficient, and its asymptotic complexity is independent of the network topology, of the number of concurrent flows, and of other network parameters. Numerical results are also presented to prove the flexibility and the power of the proposed methodology, as well as to validate it against detailed simulations.

These results provide insight into the implications of using TCP as a building block of complex systems, such as the use of TCP as the transport technology in the design of an application-level overlay.

Not many techniques exist for the design of packet networks or for evaluating the performance of systems operating on top of these networks. Designers still often rely
on the approaches devised in the 70’s, that assume Poisson packet flows, exponential packet lengths, and independence. In addition, they consider the average packet delay as the only target of network design, which is not easily related to the end-user perceived performance.

For this reason, in recent years, the development of analytical models of TCP has become a hot research topic. Researchers examined the throughput achievable by long-lived connections [7, 10, 47], and, even more importantly, the completion time of short-lived connections [15, 42, 51, 52]. Several papers proposed methods to obtain the average completion time of short-lived TCP connections (a concise overview of previous work is presented in the next section). However, the variability of QoS requirements for different types of services, hence different types of TCP connections, require the computation of more sophisticated performance metrics, such as completion time distributions, and quantiles of the completion time for TCP connections that need to transfer a given number of segments.

In what follows we present an efficient analytical technique for the computation of completion time distributions based on open multiclass queuing network (OMQN) models of the TCP protocol [23–27].

An OMQN model is a stochastic representation of the finite state machine (FSM) description of the TCP transmitter behavior, and provides a precise model of the TCP dynamics. It requires as input the average packet loss probability, and the average round trip time for the TCP connections considered. The OMQN model decouples the description of the protocol behavior from the description of the network behavior, so that the above two parameters can be obtained in several different ways: i) measured over an actual network or an experimental setup, ii) obtained from simulation experiments, iii) estimated with an analytical model of the underlying IP network; iv) set by the network designer to find planning requirements and operating points.

This flexibility allows OMQN models to be used in any networking scenario. For instance, in designing a mixed wired and wireless network, the design objectives can be divided between the wired and the wireless part. From an analytical perspective, decoupling the TCP model from the network model enables to tackle complex, heterogeneous scenarios that are not tractable by more structured approaches where the behavior of TCP and network are modeled jointly. See for instance the joint modeling of TCP with AQM routers in [38, 43, 48].

---

1 Quantiles are values which divide the distribution such that there is a given proportion of observations below the quantile. Formally, given \( p \in [0, 1] \), the \( p \) quantile of a random variable \( X \) is any number \( x \) such that \( \Pr(X < x) \leq p \leq \Pr(X \leq x) \).
The availability of a model to compute completion time distributions offers advantages, regardless of how the average packet loss probability and round trip time are obtained.

- It can be argued that, if the loss probability and round trip time are obtained by measurements, completion time distributions can also be measured (although this is significantly more complex). However, the advantage of the model described here lies in allowing “what-if” analysis to assess the effectiveness of various changes in the system.

- If the loss probability and round trip time are obtained by simulation, then the completion time distribution can also be estimated from the same simulation experiments. However, CPU times needed to obtain reliable estimates for distributions from simulations are exceedingly long. The technique we propose allows simulation to be used only to obtain parameters that are (relatively) easy to estimate accurately. Completion time distributions and their quantiles can then be computed analytically. The overall procedure is semi-analytic: some system parts are accurately modeled analytically (in our case TCP), while the parts that are too complex (non-linear, heterogeneous, etc.) are simulated.

- Finally, the completion time distribution computation can be embedded in a complete OMQN based analytical approach, operating as a post-processing procedure once the TCP and network models have been solved (see Sect. 2.3 and [23, 26, 27] for further details).

We emphasize here that the contribution of this work is neither the OMQN model of TCP (which was already presented in the literature [23, 27]), nor an IP network model, rather a methodology to derive completion times quantiles and distributions. We deem this a major step forward in analytical modeling for performance evaluation of end-to-end protocols, like TCP, and its application to multiservice networks. Given the close relationship between the OMQN and the FSM of a protocol, the extension of the presented technique to protocols different from TCP is straightforward.

The rest of this chapter is organized as follows. Section 2.2 provides a concise overview of previous work on the estimation of short-lived TCP connection completion times. Section 2.3 briefly describes the open multiclass queuing network model of TCP. Section 2.4 introduces our technique for the computation of the completion time distribution of short-lived TCP connections. Section 2.5 presents numerical results both for the validation of the analytical approach and for the demonstration of the flexibil-
ity and power of the proposed methodology. Finally, Section 2.6 contains our closing remarks.

2.2 Previous Work

A vast literature exists on analytical models of TCP, thus we cannot provide here a comprehensive overview of previous contributions; rather, we mention some of the studies that addressed the estimation of the completion time of short-lived TCP transfers, particularly those that adopt techniques similar to the one used in this Chapter.

A large number of papers tackled the estimation of the steady-state throughput of infinite (or greedy) TCP flows (those that are today often called elephants), progressively introducing more and more refinements [7, 40, 47]. However, throughput alone is insufficient to characterize the behavior of some classes of end-user applications, and is surely inappropriate for the characterization of the QoS offered to short-lived TCP flows (those that are today often called mice), whose performance is dominated by startup effects, such as connection establishment and initial slow start.

In [15], the authors extend the steady-state model of [47] to describe the behavior of finite TCP flows. In particular, they account for the connection establishment phase (whose duration distribution is estimated), and provide an approximate analysis of the first slow start phase, assuming correlated losses. Then, in order to obtain the time necessary to transfer any data remaining after the initial slow start phase, they resort to the steady-state throughput formula of [47]. Combining results, they obtain closed-form expressions for the average completion time of finite TCP flows, given as input parameters the TCP connection transfer size in bytes or segments, the TCP connection round trip time and packet loss probability.

The same approach was revisited and significantly extended in [51, 52], using a more accurate model to describe the evolution of the congestion window during the initial slow start phase. In particular, the authors provide a very detailed analysis of all cases that can occur up to the first loss event and the corresponding recovery phase. In the case of multiple losses, they compute the conditional mean transfer time for a given number of loss events, assuming that losses are uniformly spaced, and that their number is binomially distributed. Both the cases of independent and correlated losses are considered, as well as the behavior of different TCP versions [51]. A few additional assumptions and some empirical approximations are necessary to obtain expressions for the average completion time of arbitrarily long flows.

In [42] the authors use a different approach to derive the average completion time
of short-lived TCP flows. Instead of dividing the transfer into different phases, they employ recursive equations that allow all the possible error patterns to be considered. In the case of i.i.d. (independent and identically distributed) losses, their results are very accurate, and their analysis is tractable up to about 10-packet TCP connections; longer connections quickly become intractable due to the exponential growth in the number of error patterns.

TCP connection completion time distributions were studied by simulation in [44]. There, the authors considered the impact of different models of loss correlation, and the effect of lost acknowledgments, suggesting that simplified Markov models could yield good approximations for the entire distribution as well as for the expectation.

In [9], Critical Path Analysis was applied to analyze packet traces obtained by tcpdump at the end-points of HTTP connections, with the purpose of understanding what fraction of the total transfer latency is due to file size, packet propagation time, network variation (e.g., queueing delays at routers), packet losses or server load. The authors also investigated the causes of variability in transfer durations, showing that timeouts are the overwhelming contributor for small and medium sized files.

### 2.3 Open Multiclass Queuing Network Model of TCP

The behavior of TCP, like that of any protocol, can be specified through a Finite State Machine (FSM) that describes the evolution of the protocol from one state to the next. Transitions between states are triggered by events (timers expiration, ACK reception, etc.). Actually, TCP is described through two FSMs, one for the transmitter and one for the receiver, but the protocol performance is driven essentially by the transmitter behavior, provided that the receiver buffer is not a bottleneck.

In [23, 24, 26, 27], the use of open multiclass queuing networks (OMQN) was proposed to develop detailed models that can accurately estimate the performance of a number of competing TCP connections. The OMQN model $Q = \{q_i\}$ comprises individual queues $q_i$ that describe the possible states of the FSM. Customers at a queue represent TCP connections in a given state. Queue service times $L(q_i)$ describe the permanence of connections in the corresponding state, and are associated with generally distributed random variables. Since there are no limits to the number of TCP connections in

\[ L(q_i) \]  

The queue service times $L(q_i)$ mainly depend on the TCP connection round trip time; hence, the accurate characterization of service times would require the knowledge of the round trip time distribution. Such distributions are rarely available, but it is possible to use constant service times calculated from the average round trip time.
any given state (the connection state is a local variable), queues have infinite servers. The transition probability from \( q_i \) to \( q_j \) is defined as \( T(q_i, q_j) \), and its value is derived from the protocol parameters and the underlying IP network characteristics. Since TCP connections are fairly independent from one another (neglecting synchronization phenomena and application-induced correlations), the arrival process in \( q_i \) \( \forall i \) is assumed to be Poisson. Hence, every queue in \( Q \) is an \( M/G/\infty \) (or \( M/D/\infty \)) queue.

The protocol behavior is completely described by the set \( \{ Q; L(q_i); T(q_i, q_j) \} \) \( \forall i, j \); however, an important piece of information is still missing for the description of a whole TCP transfer: the backlog of the connection, i.e., the residual number of segments to be transmitted to complete the transfer. This information is associated with customers (i.e., TCP connections) through a class \( c \in [0, N] \) where \( N \) is the number of segments to be transferred when the connection opens. A new customer (TCP connection) enters the system in queue \( FE_1 \) (identical for all connections) describing the initial state for all TCP connections, with class \( c = N \). The class \( c \) of a customer is decremented whenever the customer visits a queue which represents the successful transmission of a segment; when a customer reaches \( c = 0 \), it enters a ‘closing queue’ \( q_M \), and then leaves the system. This latter queue can for instance represent the half-close procedure of TCP. Any connection in the system is completely described by the state \( s = (q, c) \).

Since queues have an infinite number of servers, the steady-state solution of the OMQN model reduces to solving the flow balance equations, hence it is computationally very light. The model can be solved either by having as inputs the network parameters, basically the packet loss probability \( P_L \) and the average round trip time \( RTT \), or by jointly solving a network model with a fixed point approximation (FPA): given \( P_L \) and \( RTT \), the OMQN provides the load to the network, which in turn returns new values of \( P_L \) and \( RTT \). The convergence of the FPA is proven in [25].

For further details on OMQN modeling, results, and FPA iterative solution, we refer the reader to the cited papers.

### 2.3.1 Customer’s Behavior in the OMQN Model

In [24, 26], average TCP connection completion times were computed using Little’s result, starting from the average number of customers (TCP connections) in the OMQN.

---

\( ^3 \) We use here the same conventions used in [24, 26] in naming queues. The number associated with the queue always indicates the current window size, while letters identify the protocol phase. For instance \( FE \) (First Exponential growth) indicates the first Slow Start, where the slow start threshold is still undefined; \( FT \) (First exponential growth Timeout) indicates the wait for a timeout to expire during a Slow Start; \( L \) a Congestion Avoidance phase (Linear growth); \( R \) the retransmission of a segment.
However, the OMQN model contains much more information that can be used, as explained in Section 2.4, to compute the distribution of the time necessary to complete the transfer of a file with given length through a TCP connection. First of all, notice that all customers (TCP connections) enter the OMQN in the same queue $FE_1$, and, after visiting an arbitrary number of queues and classes, exit the OMQN from the same state $s = (q_M, 0)$.

Fig. 2.1 depicts a small portion of an OMQN model, with the evolution of two extremely simple TCP connections. White boxes are queues in $Q$, solid lines represent possible transitions between states. $U_x$ and $V_x$ (in shaded circles) are two customers (TCP connections), $x$ being their current class. Dashed lines indicate the customer movements over the OMQN model.

$U$ enters the system in class 4 (with 4 segments to transfer). Let’s assume that $U$

\[\text{Fig. 2.1: Samples of connection evolution within a simplified TCP OMQN model}\]
does not experience any loss. One segment is transmitted from $FE_1$, and the correct reception of the first ACK after one RTT triggers the transition to $FE_2$, where two segments are transmitted. Receiving the ACK of the second segment (after another RTT), enables $U$ to transmit the fourth segment in queue $FE_3$, thus reaching class 0, moving to $q_M$, and finally leaving the OMQN.

$V$, instead, enters the system in class 3 (with 3 segments to transfer), and we assume in this case that the second segment is lost while traversing the IP network. The transmission of the first three segments proceeds as for $U$; however, since the ACK of the second segment is not received, the retransmission timer of $V$ expires. $V$ goes to queue $FT_2$ and retransmits the segment from queue $R_1$. Notice that the class is decremented by one going from $FE_2$ to $FT_2$, since only one of the two segments transmitted from $FE_2$ is lost. If the segment is retransmitted successfully, a cumulative ACK (we do not consider the delayed ACK option) allows the customer class to reach 0, so that $q_M$ is entered and $V$ can leave the OMQN, having completed the transfer.

A complete OMQN model of TCP NewReno connections is shown in Fig. 2.2 under the assumption that the maximum window size is 10 segments (this is the smallest value that allows us to show a full model; larger window sizes only produce an increase of the number of queues, but no new types of queues are added). Queues in OMQN are arranged in a matrix pattern: all queues in the same row correspond to similar protocol states, and all queues in the same column correspond to equal window size. Transitions arriving to the closing queue $q_M$ are not shown in the figure to avoid cluttering of the graph. The OMQN models TCP dynamics precisely accounting for startup effects, timeouts due to consecutive losses and the effects of multiple segment losses. The fast recovery algorithm is also modeled. For more details, the reader is referred to [23, 24, 26, 27].

### 2.4 Computation of the Completion Time Distribution

Depending on the occurrence of losses, the life of an $N$-segment TCP connection may exhibit many different behaviors, and may correspondingly have different durations. Each behavior is represented by a possible path followed by a customer through the OMQN. Therefore, by properly exploring and analyzing the set of possible paths in the OMQN, we can compute the distribution of the completion time of an $N$-segment TCP connection.

In particular, we associate two values with each path: the probability that a customer follows the path, and the delay experienced along the path. We then combine
Figure 2.2: The OMQN model of TCP NewReno
probabilities and delays of all paths to derive the completion time distribution.

Since a path represents the dynamics of a TCP connection, it is convenient to describe it with the ordered list of states visited by the customer representing the connection. More formally, define path \( i \), denoted by \( h_i \), as the ordered list of the \( n_i \) states visited by a connection during its lifetime,

\[
h_i = < s_1, s_2, \cdots, s_{n_i} >
\]

where \( s_k = (q_k, c_k) \) is the \( k \)-th state visited by the connection. In general, different paths have different numbers \( n_i \) of visited states. Denote by \( \mathcal{H} \) the set of all possible paths, \( \{h_i\} \). Due to the structure of the queuing network, the set \( \mathcal{H} \), as well as the path lengths \( n_i \)'s, are finite. In fact, a customer can never visit the same queue in the same class more than once: it either will never visit the same queue again, or will visit the queue after at least one successful packet transmission, which yields a decrease in the class number. As already mentioned, we associate with path \( h_i \) the following values:

- the probability \( P(h_i) \) that the customer follows path \( h_i \);
- the delay \( D(h_i) \) incurred by the customer to follow path \( h_i \) in the queuing network.

The probabilities \( T(s_k, s_{k+1}) \) that describe the customers’ transition from queue \( q_k \) to queue \( q_{k+1} \) while the customer class changes from \( c_k \) to \( c_{k+1} \) are all independent. Because of this independence, the probability \( P(h_i) \) can be computed as follows:

\[
P(h_i) = T(s_1, s_2) T(s_2, s_3) \cdots T(s_{n_i-1}, s_{n_i}) = \prod_{k=1}^{n_i-1} T(s_k, s_{k+1}) \quad (2.1)
\]

The delay associated with path \( h_i \) is given by:

\[
D(h_i) = \sum_{k=1}^{n_i} L(s_k) \quad (2.2)
\]

where \( L(s_k) \) is the latency (or service time) of queue \( q_k \). \( D(h_i) \) represents the \( N \)-segment connection completion time, given that the connection behaves according to path \( h_i \).

Since all the paths \( h_i \in \mathcal{H} \) are mutually exclusive, the cumulative distribution function \( F_X(t) \) of the random variable \( X \) which represents the \( N \)-segment TCP connection completion time can be computed as:

\[
F_X(t) = P\{X \leq t\} = \sum_{i \in \mathcal{H}_t} P(h_i) \quad (2.3)
\]

with \( \mathcal{H}_t = \{h_i|D(h_i) \leq t\} \subset \mathcal{H} \).
2.4. Computation of the Completion Time Distribution

Unfortunately, since the cardinality of $\mathcal{H}$ follows an exponential law with the number $N$ of segments to be transferred over the connection, the exact computation of $F_X(t)$ is not feasible in most realistic cases, i.e., for connections longer than few segments. Following a deterministic approach, the authors in [42] set a maximum of 10 segments per connection.

Since the exact computation of $F_X(t)$ is not feasible, we are interested in developing a general methodology for the approximation of $F_X(t)$ reducing the complexity of the computation, while still closely approximating the distribution, both for small values of $t$, and for its tail as $t \to \infty$. Moreover, it should be possible to reduce the approximation error with an arbitrary precision.

The basic idea of the procedure is the following. We approximate the random variable $X$ with a variable $X'$ that is distributed on a smaller number of possible values, i.e., we discretize the values that $X'$ can assume. Then, instead of computing $F_X(t)$ as in (2.3), after having derived all the path probabilities and delays, we compute $F_{X'}(t)$ while we are still generating the possible paths. In particular, while we are discovering paths, each transition from one queue to another contributes to the computation of the probability and of the delay of those paths that include the transition.

Let the random variable $Y_s$ denote the first arrival time to state $s$, that is, the time spent by the TCP connection from set-up until state $s$ is entered. $Y_s$ is a discrete random variable that can assume values in $\mathbb{R}$, exactly like $X$. We can write the recursive expression for its probability mass function:

$$P\{Y_s = t\} = \sum_{r \in S} P\{Y_r = t - L(q_r)\}T(r, s)$$

(2.4)

where the transition probability $T(r, s)$ from state $r$ to state $s$ is equal to zero if the transition is not possible. If from state $r$ a transition to $s$ occurs with probability $T(r, s)$, we deduce that with this probability the delay is equal to the delay to reach $r$ plus the time spent in queue $q_r$, i.e., $L(q_r)$. We accordingly update the distribution of $Y_s$. By so doing, the distribution of $Y_{s_0}$, where $s_0 = (q_M, 0)$ is the end state of all paths, coincides with the distribution of $X$.

The starting point of the recursive procedure is the definition of the distribution for state $s = (FE_1, N)$, which is deterministic with 0 delay ($P\{Y_{FE_1, N} = 0\} = 1$), since the entry point into the queueing network is always queue $FE_1$ in class $N$.

\footnote{As it was previously noted, a customer can never visit the same state again in the same class, so the first arrival to a state is also the last one. Not all paths reach a given state; for these paths, the first arrival time can be defined as being a large finite arbitrary value, say $\Theta$.}
It is worth noticing at this point, that if $X$ and $Y_s$ assume values in $\mathbb{R}$, the complexity of the computation of $F_X(t)$ and $F_{Y_s}(t)$ is equivalent. Discretizing the support of the random variable, instead, leads to the merging of large numbers of paths, which decreases the complexity of the computation of $F_{Y_s}(t)$ with the recursive approach. The dynamics of TCP, which evolves in multiples of the round trip time, and the structure of the OMQN model, with a low connectivity degree, also help the process, since paths tends to cluster together around integer multiples of the round trip time.

Let us now focus on the discretization of the values $Y_s$ can assume, which is a critical aspect of the recursive procedure in (2.4). Rather than discretizing the support of $Y_s$ a-priori, we chose to dynamically adjust the set of discrete values of $Y_s$ during the recursion. Suppose that, as shown in the upper part of Fig. 2.3, at a given step of the recursion the probability mass function (pmf) of $Y_s$ is represented by a set of values $t_i$, with probability $P_i$. Suppose also that the transition from $r$ to $s$ adds a new element to the pmf of $Y_s$, which corresponds to abscissa $t^*$ with probability $P^* = P\{Y_r = t^* - L(q_r)\} T(r,s)$. As shown in the lower part of Fig. 2.3, we have two possibilities.

1. introduce a new abscissa $t^*$ in the pmf, with probability $P^*$ (lower left part of the figure).

2. look for the abscissa $t_k$ closest to $t^*$, and update the pmf by moving $t_k$ to $(P_k t_k + P^* t^*)/(P_k + P^*)$, associating with the new abscissa a probability $P_k + P^*$.

The choice between the introduction of a new abscissa in the discretization of $Y_s$ (case 1) or the simple update of the pmf (case 2), is based on the ratio between $t_k$ and $t^*$. A new abscissa is introduced if,

$$\left| \frac{t_k - t^*}{t^*} \right| > R \quad (2.5)$$

By basing the decision on the above ratio criterion, abscissas are allowed to range over different time scales, and tend to be equally spaced in a logarithmic scale. This is an interesting feature of our approach, since we are interested in accurately estimating the tail of the TCP connection completion time distribution. Notice also that the described discretization procedure also nicely fits the distribution shape (which is typically monotonically decreasing), making the discretization less dense where the probability mass is small. Clearly, the smaller the parameter $R$, the more accurate the discretization of $X$. As $R \to 0$ the approximating distribution converges to $F_X(t)$, thus ensuring that the error can be made as small as desired.

The choice of $R$ is easy, since it represents, approximately, the inverse of the number of approximating points that are used for each decade of the abscissa. The impact of
2.4. Computation of the Completion Time Distribution

Figure 2.3: Sketch of the recursive procedure to derive the probability mass function of $Y(s)$

Figure 2.4: Impact of $R$ on the distribution of the TCP connection completion time

$R$ on the method’s precision can be observed in Fig. 2.4, which shows the cumulative distribution function (CDF) of the completion time of 100-segment TCP connections in a single-bottleneck network under load 0.85. With $R = 0.1$ we can observe a stepwise behavior in the CDF, due to a too coarse discretization, while with both $R = 0.01$ and $R = 0.001$ the CDF becomes smooth, and the discretization is acceptable. The same effect of $R$ can be seen for the tail of the distribution on the right hand plot of the figure,
which shows the complementary CDF in log-log scale.

Similar behavior was observed also in more complex scenarios; therefore, we selected the value $R = 0.01$ for the derivation of numerical results that will be discussed in Sect. 2.5.

The speed of the algorithm can be further improved by dropping some of the abscissas from the distribution of $Y_s$. Instead of always introducing a new abscissa in the first case, we choose to introduce it only if the threshold $P^* > P_{\min}$ is met. Since $P^*$ aggregates the probability of all the paths reaching state $r$ with a delay near $t^* - L(q_r)$, the cut-off criterion ensures that no significant point can be dropped because it is a sum of a large number of small probability paths.

For the derivation of numerical results we selected the value $P_{\min} = 10^{-15}$ which resulted in negligible errors, namely the total probability of all the dropped paths was always less than $10^{-10}$.

Summarizing, we can say that the proposed method is a mixture of three techniques: discretization of the time scale, aggregation of paths reaching a state in nearly the same time, and dropping irrelevant paths from the calculation. Fig. 2.5 shows the pseudocode of the algorithm used to compute the completion time distribution. Note again that the algorithm is iterative: it relies on the fact that the class number decreases monotonically, and that a customer can never visit the same queue in the same class more than once.

Computational complexity is small and allows the exploration of realistic cases. Fig. 2.6 shows execution times as a function of flow length for the case when the model is jointly solved with a network model using a fixed point approximation. First, a simple network model (for details of the network model, see the single bottleneck scenario in the next section) was solved together with the OMQN model for its steady-state using FPA. Once the steady-state solution is known, completion time distributions are calculated for the TCP streams. It can be seen that execution times scale almost linearly for both phases of the calculation.

We stress once more that the computation of these two phases are sequential, and their calculation times are independent of each other. A complex network model might increase the computation time of the FPA, however, once this is solved, the computation of the completion time distribution of TCP connections of a given size on a given path remains the same. Therefore, we can say that the complexity of the calculation of the completion time distribution does not depend on parameters of the network model such as the network topology or the number of concurrent flows.

Comparing the numerical approach to the ns-2 simulations we used for validation, the difference in CPU time ranges between 3 and 4 orders of magnitude.
2.5 Validation and Results

The objective of this section is twofold. First, we validate our model by comparison against simulation results, thus proving that the model is able to provide extremely accurate results. Second, by using only the analytical approach and taking advantage of its computational efficiency, we extensively study the QoS perceived by TCP users.

Figure 2.5: Pseudocode of the iterative algorithm for the computation of the completion time distribution
in terms of completion time quantiles. We therefore show that the proposed analytical method can be very helpful in the context of QoS-aware system design, allowing the determination of the system parameters that yield a specified maximum completion time to a given percentage of TCP connections of any given length.

### 2.5.1 Model Validation

In order to validate the technique, we compare its results against simulation experiments under various network scenarios. As a simulation tool we employ the *ns-2* simulator [6], which has become the standard for the validation of analytical models and performance improvements of TCP [13, 15, 27]. Throughout this section we use the NewReno version of TCP, although the method can be applied to other versions of the protocol as well.

Evaluating confidence intervals for distributions is not a trivial task. In order to obtain an indication of the result’s reliability, we build an indication function

$$I_{X_i < x} = \begin{cases} 1 & \text{if } X_i < x \\ 0 & \text{otherwise} \end{cases}$$

where $x$ is the (discretized) time variable and $X_i$ is the random variable.

For each value of $x$ the average of these indicators:

$$F^*_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{X_i < x}$$

can be used to estimate the confidence interval invoking the Central Limit Theorem, since the indicators are i.i.d. by construction. By so doing confidence intervals of the whole distribution can be calculated. Results referring to simulations reports three
2.5. Validation and Results

Separate curves with the same graphic tract: the middle one is the average value, while the other two (thinner lines) define the confidence interval with a confidence level of 99%.

**Single bottleneck scenario**

We first consider a simple network scenario consisting of a single bottleneck link with capacity $C = 10$ Mbps. The one-way propagation delay is 40 ms. A gentle RED queue management scheme is adopted, with low and high thresholds equal to 10 and 80 segments, respectively, with loss probability at the high threshold equal to 5%. The buffer size is 512 segments. The network is loaded by TCP connections, all of which transfer $N_s = 100$ segments, the segment size is constant and equal to $S_p = 1000$ bytes. TCP connections are generated according to a Poisson process, whose rate $\lambda$ is such that the nominal link load $\rho_n$ is equal to 0.9. TCP maximum window size is 64 segments, the tic is 0.5 s and the initial timeout is set to 6 s. The nominal load is defined as

$$\rho_n = \frac{\lambda \cdot N_s \cdot 8S_p}{C}$$

where $C$ is the link capacity in Mbit/s. Computing the actual link load, including retransmissions, can be done only a-posteriori, knowing the drop rate. The scenario is sketched on Fig. 2.7.

The cumulative distribution function (CDF) of the TCP connection completion time is shown in Fig. 2.8. The solid line reports simulation results, the two dashed lines refer to the analytical approach and differ in the way the round-trip time and segment loss probability are computed.
Figure 2.8: CDF and CCDF of the completion time; Single bottleneck, homogeneous connections with $N_s = 100$, $\rho_n = 0.9$; Analysis ($\text{RTT}=134\text{ ms}$, $P_L = 1.64\%$) and simulation ($\text{RTT}=132\text{ ms}$, $P_L = 1.61\%$).

For the results shown in long-dashed style, the analytical model of the IP network is used to derive the average round-trip time and segment loss probability\footnote{The iterative algorithm presented in \cite{26} was used to calculate these values. This way, the use of the OMQN model is twofold: the steady-state solution is used in the iterative process to obtain averages, then its dynamic properties are used to determine the distribution.}, for the short-dashed line, instead, the estimates of the round-trip time and loss probability are computed by simulation\footnote{We use the same simulation as for the validation curve, although a much shorter simulation would be enough to determine these average values.}. Showing both of these curves has two reasons:

- It allows the separation of the inaccuracy due to the IP network model from the inaccuracy in the calculation of the completion time distribution;
- It demonstrates two different uses of the completion time distribution calculation method. In the first case a network whose topology, equipment and traffic is known could be evaluated from the point of view of TCP quality with an all-analytic approach. In the second case, on the other hand, the environment in which TCP works is not important, it can be seen as a black box, where only the loss probability and the round trip time is known from some sources.

The behavior of the tail of the distribution can be observed on the right hand part of Fig. 2.8 which shows the complement of the CDF (CCDF) in logarithmic scale.
peculiar behavior of the curve, with points where the slope changes abruptly, is due to TCP initial timeouts. The first peak around 7s corresponds to the following case: after the first segment is lost, the 6s initial timeout expires and the retransmission of the initial segment is successful. The second peak is around 19s. In this case, the first segment transmission fails for two consecutive times and the backoff mechanism applies. Despite the low computational cost, the analytical approach is able not only to catch the shape of the distribution, but also to provide extremely accurate results even for small values of the probability. On the contrary, simulations require a considerable amount of computation time, especially when, as in most of the cases of interest, low probabilities are considered.

The probability density function (PDF) of the TCP connection completion time is shown in Fig. 2.9. Again, while small probability parts of the PDF, of the order of $10^{-5}$ or smaller, are quite costly to obtain by simulation, the analytical approach provides reliable estimates even for extremely low values with no additional computational cost. We simulated 1.125 millions flows in order to obtain the curve in Fig. 2.9 (we plot a point if we have at least 5 samples), and it took 3 hours. The analytical results were computed in less than 2s.

We now consider shorter TCP connections that transfer only 10 segments, sharing the same single-bottleneck network described before. The CCDF is in this case shown in
Figure 2.11: Two bottlenecks scenario, non-homogeneous connections. Setup (left) and simulation results (right). Flows from router 2 to router 1 (RTT=47 ms, PL = 0.7%) and from router 3 to router 1 (RTT=95 ms, PL = 2.7%).

Fig. 2.10 Again, analytical estimates are very accurate. However, the curve reporting simulation results exhibits a smoother behavior. This is due to the deterministic round-trip time assumption introduced in the model. Consider for example the steep decrease of the curve when the delay is about 450 ms: it corresponds to the case in which no segment losses occur. While by simulation the no loss case translates into values of completion time varying between 350 and 800 ms due to the variability of the round-trip time, in the analytical model only one path corresponds to the no loss case, so that a deterministic completion time is predicted.

By comparing this plot to the one in Fig. 2.8 which refers to 100 segment connections, it can be observed that behaviors are similar, but shorter connections lead to steeper changes in the CDF. In fact, for long connections, a large set of possible paths combine to form the CDF and this makes the curve behavior smoother.

Multiple bottleneck scenario

In order to validate our approach in a more complex network environment, we consider the 3 routers scenario sketched on the left hand part of Fig. 2.11. The traffic used is also more realistic, it consists of non-homogeneous TCP connections: 50% of the generated connections contain 10 segments, 40% contain 20 segments, and the remaining 10% of the connections are 100 segments long. Of course, connections crossing different paths perceive different round-trip time and loss probability. The right hand side of
2.5. Validation and Results

Figure 2.12: Analysis and simulation. CCDF. Two bottlenecks scenario, non-homogeneous connections. Connection length $N_s = 100$ (left) and $N_s = 10$ (right).

Fig. 2.11 shows, by simulation only, the reason why we are interested in the completion time distribution of different TCP flows in various network conditions: differences in network conditions induce significant differences in completion time distributions. The plot shows flows traversing routers 2 and 1 (labeled “Flow 2 → 1” in the Figure) and flows traversing the three routers (labeled “Flow 3 → 1” in the Figure). Curves for flows traversing routers 3 and 2 are omitted for simplicity. Due to larger loss probability and round-trip time, the completion time for connections crossing the three routers tends to be significantly longer than the completion time of the other flows. The difference caused by the amount of data to be transferred is smaller than the difference caused by the path, specially when the distribution tail is considered (only connections of lengths 10 and 100 segments are shown for clarity).

The validation of the analytical approach in this more complex scenario is shown in Fig. 2.12. These results confirm the accuracy and flexibility of the model. Simulation based analysis is limited due to required time, thus the accuracy of the analytical model opens new possibilities, as discussed in Sect. 2.5.2.

Mixed UDP and TCP traffic

As the last part of the validation, to demonstrate the model’s ability to handle a wide range of loss probabilities and even more realistic network traffic, we show a single bottleneck network overloaded with TCP and UDP traffic. In this scenario 2 Mbps UDP traffic is generated by CBR sources, while the nominal link load of TCP connections is
Figure 2.13: CDF and CCDF. Single bottleneck scenario, mixed UDP and TCP traffic. Connection length \( N_s = 10 \). Analysis and simulation (RTT=175 ms, \( P_L = 29\% \)).

set to 0.85. This overload situation resulted in a very high drop rate of 29\%. Fig. 2.13 clearly shows that the analytical model could handle even this large drop rate with good precision. (The analytical model of the IP network we used for the fixed point algorithm does not support UDP traffic, therefore the curve “using NW model” is not shown).

### 2.5.2 Further Results

Besides allowing a deep understanding of the protocol dynamics, the analytical model has two major advantages over simulation. First, it is much more efficient; second, the computational cost is almost insensitive to the network scenario, the values of the network parameters, and the order of magnitude of the probability of interest. Therefore, the analytical model can be efficiently used in the context of planning and performance evaluation of IP networks.

Fig. 2.14 reports a 3-D plot of the mean of TCP connections’ completion time as a function of the connection length and the segment loss probability, given a 50 ms round-trip time. Of course, large values of loss probability and connection length correspond to large values of the mean completion time. As known from measures and empirical models, the average completion time grows linearly with the connection length, while it grows with an hyperbolic law with the segment loss \( P_L \).

Fig. 2.15 shows 0.95 and 0.99 quantiles in the same situation. It is interesting noticing that curves describing delay quantiles are not “scaled versions,” of the average delay, but tend to have different shapes, with a step-wise behavior that is more pronounced as
2.6 Summary

In this chapter we have proposed and discussed a path analysis technique applied to open multiclass queuing network (OMQN) models of the TCP protocol that can be used to obtain reliable and accurate estimates of the distributions and quantiles of the completion time of short-lived TCP connections.

The OMQN models of TCP require as input only the packet loss probability, and
Chapter 2. Performance of TCP-based Overlay Tunnels

Figure 2.15: Analytical results: 0.95 (left) and 0.99 (right) completion time quantiles as a function of the connection length $N_s$ and of the segment loss probability $P_L$.

the average round trip time for the TCP connections being considered. These can be either obtained through measurements and simulations, or estimated with an analytical model of the underlying IP network, that works in conjunction with the OMQN model of TCP. In all cases the same path analysis post-processing can be applied.

Numerical results prove the flexibility and the accuracy of the proposed methodology, whose computational complexity is small and does not depend on network load or topology. Several interesting features of the completion time distribution, that cannot be simulated due to prohibitive simulation times, can be highlighted with the proposed approach.

The ability to accurately estimate quantiles of TCP connection completion times provides a very important support to the design of systems relying on TCP as their transport or tunneling protocol.
Chapter 3

IPsec-Based Anonymous Networking

Protecting users’ privacy is becoming one of the rising issues for the success of future communications. The Internet in particular, with its open architecture, presents several threats to the right of users for protecting their personal and sensitive data.

One fundamental building block of privacy-preserving communications is protecting the communicating parties’ identities, or, as it is commonly called within the research community, anonymous networks (ANs). An AN prevents external observers as well as the network to have access to communicating partners’ identities and addresses. We propose a novel architecture to realize ANs that distinguishes itself from other architectures by composing data-path functionality entirely from standard IPsec elements. We call this new architecture IPpriv. After explaining the rationale and discussing possible alternatives, we present a working prototype implementation and its experimental performance comparison with application level solutions.

3.1 Background and Privacy Protection

One of the challenges for the “network of the future” is improving the protection of users against frauds and intrusions, including the provisioning of suitable means to protect personal and sensitive information. In some cases preserving privacy implies the right to be anonymous, including in this term pseudonymization techniques, where the identity of a person is disguised, but can be revealed to selected authorities (e.g., lawful investigators authorized by a judge). One typical example is voting, where revealing the identity of voters is always the first step toward election frauds.

Until very recently, it was deemed that secrecy of data communications (i.e., strong enough cryptography) was enough to preserve users’ privacy. Anonymity of both addresses and users was not even conceived. Now, instead, privacy protection is recognized as one of the functions that a telecommunication network should provide to its customers and users.
Due to the addressing space of the Internet, source and destination IP addresses can be seen all along the communication path, revealing the location (through IP geolocation) as well as the identity (through reverse DNS; by linking different actions initiating from the same IP address; or by contacting the ISP who assigned the address) of the communicating parties. Once the identity is uncovered, privacy can be compromised. Widespread use of customer databases and activity logs, and the recent evolution of data mining and linkage techniques make it possible to obtain private and sensitive data analyzing traffic flows and communication end-points.

Privacy in the “Internet” was first discussed by David Chaum introducing the concept of anonymous communications, but after this initial discussion the topic remained confined in small communities for a long time. The concept, which was originally thought for anonymous e-mail only, was later extended to low-latency communications by application level overlay systems such as Tor, Tarzan or Freedom. Low-latency in this context means that the communication is not based on storing and forwarding entire application level messages in intermediate nodes, but rather small information units are forwarded limiting end-to-end delays to seconds.

Our contribution is showing that IP address protection can be provided within the network layer itself, namely within the IPsec framework, without the need of building cumbersome and non-scalable application-level overlays as any other AN solution does.

We implemented the proposed architecture using standard IPsec and routing configuration tools available in Linux 2.6. Measurements on an experimental testbed show significant performance advantage of our architecture compared to the Tor architecture, which is the de-facto standard in AN, with thousands of overlay routers in operation and over 200,000 users.

3.2 Principles of Anonymous Networking

Anonymity at the network level means not revealing the address of one end-point to the other side, and preventing any third party observing the network from understanding who is communicating with whom.

Referring to Fig. 3.1, the goal of an AN is protecting Alice’s identity from Bob, as well as the fact that they are communicating with each other, from Mallory, an external attacker. We call a node of an AN a mix. A mix preserves communication privacy, including anonymity, and does this by mixing traffic belonging to different end-to-end connections.
3.2 Principles of Anonymous Networking

Figure 3.1: Protecting anonymity in face of the destination and of an external attacker with a favorable observation point where several links and routers can be controlled.

The threat model we consider is similar in scope to many low-latency ANs, it comprises most of the modern traffic analysis techniques. To provide sender-receiver unlinkability at least one of the sender or the receiver addresses must be hidden at any observable point of the network. This way, Mallory, even if being near the source or the receiver, can only see that one of Alice or Bob is communicating with someone, but is not able to understand who the communicating parties are.

A global adversary with unlimited resources to monitor links and routers and correlate information (or with initial clues about communicating pairs) can still uncover who is communicating with whom by matching packet size, packet timing, or flow characteristics. Protection against such an attack is possible using traffic flow confidentiality techniques, such as the IPsec extension introduced in [33] or any other suitable proposal, which are however outside the scope of this Chapter.

The first system designed to provide ANs was [16], which defined the founding principles of anonymous communications, summarized as follows:

P.1 Un-traceable delivery: messages are delivered through a chain of mix nodes, which enforce that no correspondence can be found between incoming and outgoing messages;

P.2 Un-traceable return address: a mechanism that allows the receiver to respond while the sender still remains anonymous;

P.3 Distributed trust: no universally trusted authority is required to keep the identity secret. Each mix knows only the previous and the next mix on the delivery path: the only trust required is that they will not collaborate to find out the
Figure 3.2: Anonymous networking realized with Onion Routing

Technically, these principles are realized using onion encapsulation (see Fig. 3.2), i.e., embedding one encrypted layer into another recursively. For this reason mixes are also called *Onion Routers* (ORs). The source selects the path of the message and encrypts it in multiple layers, using the public-keys of the selected mix nodes one after another, including the address of the next mix node in each layer of encryption. Each mix “peels off” one layer, delivering the payload to the next mix in the chain. A chain of two nodes is enough to provide anonymity, but three or more are normally used to improve protection by reducing the possibility that mixes collaborate.

To provide low-latency, high throughput anonymous service, public-key encryption is computationally too demanding. To overcome this limit, faster symmetric key cryptography must be used, dividing communication into two phases. When a node wants to use AN services, first, in a circuit-setup phase, symmetric keys and circuit identifiers are exchanged with public-key cryptography with each OR of the chain. Later, during data transmission on this circuit, faster symmetric keys are used for the onion encapsulation.

These kind of anonymous circuits are normally called ‘telescopes’ because the source encapsulates all headers related to tunnels toward ORs one into the other, and each OR decapsulates one header, so that the overall structure of nested tunnels is thicker at the source and becomes thinner toward the destination. In what follows we use the terms circuit and telescope interchangeably.

Implicit in the notion of telescope is the use of source routing: it is the source that defines which mixes are to be used. Source routing is not the most efficient routing solution; however, at the state of the art no other viable solution to route packets and protect privacy has been found.
The topological structure of the whole AN can be seen as an overlay of mix nodes on the IP network, serving a large number of telescopes. Mix nodes are interconnected by overlay links, implemented as tunnels crossing multiple standard IP routers. These long-term overlay tunnels are different from nested tunnels of telescopes, and provide some basic functionalities to organize the network. They convey information among mix nodes forming the edges (or links) of the anonymous network. They provide encryption to make packets belonging to different circuits indistinguishable from each other. Finally they can provide traffic flow confidentiality to prevent statistical analysis and confirmation attacks.

Existing ANs [12, 21, 22] make different design choices for the implementation of overlay tunnels, but all of them work at the application level either by using TLS tunnels or proprietary solutions. For implementing the nested tunnels of telescopes, they all use proprietary solutions. All systems are 'all-in-one' solutions, making it difficult to evolve them and to make them highly performant and widespread in use. In [34] we discuss the limitations and performance impairments (from an architectural/theoretical point of view) of using transport (or application) level tunneling.

Summarizing this Section, an AN can be realized with a proper combination of the following techniques.

A.1 A procedure to open and maintain secure tunnels, both for overlay links and for nested tunnels of telescopes; the telescope tunnels should hide end-point addresses.

A.2 A path selection mechanism to choose the mixes forming the telescope.

A.3 An anonymous packet forwarding mechanism that enforces the specified path, both in the forward and in the reverse direction.

A.4 A mechanism that allows any host (or generic IP node) to communicate over the anonymous overlay network through anonymity gateways and exit nodes.

A.5 A telescope setup mechanism to distribute (tunnel’s) state information to mixes of the selected path.

A.6 A management functionality to distribute topology information both to mixes (in order to set up overlay tunnels) and to clients (to provide information for path selection), and to distribute public keys and similar maintenance information.
3.3 IPpriv: IPsec-Based Anonymous Networking

IPsec natively supports configurations where several layers of tunnels are nested in each other \[32\]. After a short primer to IPsec, we show how this nested tunneling capability, together with additional properties of IPsec and related signaling protocols can be manipulated and used to fulfill the points A.1, \ldots, A.5 that emerged at the end of Sect. 3.2 as critical requirements to implement AN. The realization of A.6 is not discussed here.

3.3.1 IPsec Primer

IPsec\[32\] is designed to provide cryptographically-based security for IPv4 and IPv6 at the packet level for all protocols that may be carried over IP (including IP itself).

The IPsec framework is a modular architecture for securing communications over IP. Essentially, it is composed of four types of components: i) security protocols; ii) cryptographic algorithms; iii) key management; and iv) negotiation protocols. Local management databases glue together the other components through the concepts of security policies (SP) and security associations (SA). Entries of the security policy database define whether packets passing the system should be protected, left unprotected or discarded; as well as the type of protection that should be applied. SPs can be established manually by a user or system administrator, but can also be managed automatically by applications such as a user-space daemon. Of particular importance to us is the following IPsec feature: SPs support the definition of several encapsulated layers of protection that are applied by the IPsec compliant system in the specified order.

The SP, in itself, only defines the need for protection. For each simplex “connection”, a security association (SA) should be created which holds symmetric keys, the label of the tunnel (SPI), and other state information needed for the actual protection. It is important to note that the meaning of “connection”, i.e., the packets that will be multiplexed in the same protected tunnel, can be configured on a wide scale, from a single TCP connection to all the IP packets passing between two nodes. SAs can be configured manually, or instantiated automatically after negotiation between the tunnel end-points, e.g. with the IKE protocol. This negotiation, in fact, can be triggered automatically if an SP requests protection for a packet, but the corresponding SA cannot be found in the local database.

IPsec provides security services directly operating at the IP packet level, in contrary to other standard (and non-standard) techniques such as SSL/TLS, SSH, etc. that work at the transport or application level. This grants three unique features to IPsec:
• besides providing end-to-end protection, a part of the connection can also be protected between security gateways by forwarding the packets in an IPsec tunnel;
• by sending the protected packet to another security gateway, IPsec can also modify the route taken by the packet through the IP network;
• layer 3 routers can natively support IPsec and act as tunnel end-points, having their SP and SA databases configured through appropriate management interfaces (e.g. the PF_KEY API[41] or the IPsec SPD Configuration MIB[8]).

In what follows, we use IPsec with the ESP (Encapsulating Security Payload) security protocol in tunnel mode. ESP applies encryption and encapsulation techniques at each IP packet in order to provide a selected subset of confidentiality, data origin authentication, connectionless integrity, replay protection or limited traffic flow confidentiality services. Once protected by ESP, an IPsec tunnel encapsulates the cyphered IP packet (including its original IP header) in another IP packet by adding a new (external) IP header with source and destination addresses of the security tunnel, as specified in the SP. After the packet is encapsulated as dictated by the SP, the packet structure becomes $IP|ESP|IP|payload$. This procedure might be repeated several times creating nested tunnels, before the packet is routed according to the external IP header towards the security gateway on the other side of the outermost tunnel. For further details, we refer the reader to [31, 32].

3.3.2 Overlay and Telescope Tunnels

Overlay Tunnels

Let OR$_{1}, \cdots OR$_{$n$} be the IPsec routers that implement the overlay. Long-term tunnels between these nodes are the edges of the overlay. Let $T$_{$O$}$_{i,j}$ be the overlay tunnel established between OR$_{i}$ and OR$_{j}$. Overlay tunnels are bidirectional, so $T$_{$O$}$_{i,j} \equiv T$_{$O$}$_{j,i}$.

The tunnels $T$_{$O$}$_{i,j}$ can be set up manually for smaller deployments, or they can be established automatically using an IPsec compatible signaling protocol (e.g., IKE or IKEv2), once certificates are deployed in the OR nodes.

Telescopes: Nested Secure and Anonymous Tunnels

Let OR$_{c_1} \cdots OR$_{$c_k$} be the nodes of the anonymous circuit $c$ between clients $A$ and $B$. We implement the circuit through $k$ nested IPsec tunnels ($T_{A,c_1} \cdots T_{A,c_k}$), each one having $A$ as one end-point and OR$_{c_i}; i = 1, \ldots, k$ as the other. These nested tunnels are also bidirectional.
Setting up the telescope described above does not provide anonymity. Packets passing from A to B are routed on the selected circuit and the IP addresses of A and B are protected from external eavesdroppers by the overlay tunnel encryption, but the address of A, as the initiator of each \( T_{A,c_i} \) tunnel, is seen by every OR\(_c_i\).

To enforce anonymity we need to masquerade the IP address of A in these tunnels. During signaling phase, the client negotiates a private IP address with each node of the circuit (\( A_{c_i} \) with OR\(_{c_i} \), etc.), and this address is used as source IP address in \( T_{A,c_i} \).

IPsec allows the creation of all the aforementioned tunnel structures setting Security Policies (SP) and creating related Security Associations (SA), thus fulfilling the requirement A.1. For details about Security Policies and Security Associations see [32].

### 3.3.3 Path Selection

As already mentioned, all ANs use source routing as path selection mechanism. The source A chooses the \( k \) nodes OR\(_{c_1} \ldots OR_{c_k} \) of the circuit randomly with uniform distribution among the available OR\(_1 \ldots OR_n\). Previous works showed that \( k = 3 \) guarantees strong anonymity protection while keeping the overhead low [21]. Once the path has been chosen, it is fixed in A’s SP database. The standard IPsec packet selector associated with this SP allows fine-grained configuration of what traffic should be anonymized and thus routed over \( c \). The following nested SP is created in A to satisfy criterion A.2:

\[
\text{selector} \rightarrow \text{PKT} \supset T_{A,c_k} \supset \ldots \supset T_{A,c_1} \supset T^O_{A,c_1}
\]

where the operator \( \supset \) means tunnel encapsulation, a per-packet (PKT) operation, and selector stands for a standard IPsec SP selector. By creating similar SPs with different selectors, several anonymous circuits can be used at the same time.

Random path selection is just the simplest possible solution. Other path selection strategies are possible [53], but details about path selections are irrelevant to our work.

### 3.3.4 Anonymous Forwarding

The system implements policy based routing in the ORs. For a given circuit \( c \) each participating OR\(_{c_i} \) installs SPs to receive and forward packets in the right direction with the necessary encapsulation/decapsulation operations, both in the forward and in the reverse path.

In the forward direction, OR\(_{c_i} \) terminates both the overlay tunnel \( T^O_{i-1,i} \) and the nested tunnel \( T_{A,c_i} \). The packet is routed toward OR\(_{c_{i+1}} \) based on the destination address of the internal IP header, and therefore it is automatically encapsulated in
The following SPs are created for each circuit:

\[(A_{ci+1}, OR_{ci+1}) \rightarrow (PKT \subset T^O_{ci-1, ci} \subset TA, ci) \supset T^O_{ci, ci+1}\]  \hspace{1cm} (3.1)

where \((S, D)\) is a selector matching IP packets with \(S\) as source and \(D\) as destination, and the \(\subset\) operator means decapsulation.

In the reverse direction, after decapsulation from the overlay tunnel, the packet is identified as belonging to \(c\) based on the unique \((OR_{ci+1}, A_{ci+1})\) address tuple. Thus, a layer of telescope encryption is added, and PKT is forwarded to \(OR_{ci-1}\) through \(T^O_{i,i-1}\). The following SPs enforce these operations:

\[(OR_{ci+1}, A_{ci+1}) \rightarrow (PKT \supset T^O_{ci+1, ci}) \subset TA, ck \subset T^O_{ci-1, ci}\]  \hspace{1cm} (3.2)

The enforcement of SPs (3.1) (3.2) in \(OR_{c1} \ldots OR_{ck}\) guarantees the anonymous forwarding in both directions satisfying A.3.

### 3.3.5 Anonymity Gateways and Exit Nodes

To make AN ubiquitous (satisfying A.4), there are two possibilities: i) extend AN services to any IP host; and ii) setup trusted anonymity gateways allowing any node to make anonymous connections. The first choice is clearly impractical, while the second one is feasible if anonymity gateways (AG) and exit nodes (EN) are implemented.

An AG is a node that carries on anonymization services for client nodes that cannot handle (or do not want to handle) anonymization by themselves. It is implemented by extending the scope of A’s SP selector to include forwarded traffic besides traffic originating from A itself. Such an AG can be placed in the default route of the client (like e.g., a home router), or can be contacted on-demand through an IPsec tunnel or using SOCKS.

An EN employs standard stateful source NAT to allow anonymous connections through the overlay toward any IP host. When \(OR_{ck}\) is an exit node, after the decapsulation of \(TA, ck\), it receives a packet with source \(A_{ck}\) destined for an ordinary node \(B\). It uses source NAT to change the source address to \(OR_{ck}\) and sends out the packet. \(B\) responds to \(OR_{ck}\) which changes the destination address based on NAT state to \(A_{ck}\), and thus policy based routing works as described before.

Fig. 3.3 summarizes packet transformations along \(c\) in the forward path, for the simplified case where \(A\) is part of the overlay and \(k = 2\). Since return path operation is the exact opposite of forward path operation, reverse path transformations can be seen reading Fig. 3.3 bottom-up.
Figure 3.3: Structure of onion encrypted packet leaving the source and subsequent packet transformations (ESP trailer omitted for simplicity); the shaded part of the packets is encrypted and thus not 'visible' to OR\(_{c_1}\).

### 3.3.6 Telescope Setup Mechanism

For setting up a telescope, shared symmetric keys should be negotiated between \(A\) and all the ORs of the path. During the negotiation, \(A\) should remain anonymous. [46] generalizes signaling that can be used for telescope setup, and the symmetric keys negotiated by the introduced signaling can be used to configure SPs and SAs through local management protocols such as the PF\_KEY API [41] or the IPsec SPD Configuration MIB [8]. In our proof-of-concept implementation pre-shared keys are used for telescope encryption for the sake of simplicity.

### 3.4 Performance Measures and Analysis

The IPsec-based AN described in Sect. 3.3 is a complex architecture with many facets and details. One of the main reasons to propose network-layer AN is the conjecture that its performance will be much better than application-level overlays. To prove this conjecture we implemented a proof-of-concept AN with the key features over Linux (2.6
kernel).

Our proof-of-concept includes all encapsulation and decapsulation procedures, management of SPs and SAs, and EN, as well as all information transfer procedures; it does not include all signaling mechanisms, since the performance during information transfer is not influenced by signaling. The implementation is fully functional and implemented in kernel space, and it is available from the authors upon request.

Fig. 3.4 describes the measurement setup. We compare the performance of Tor and IPpriv. To evaluate the “cost” of privacy we choose two measures: web page download time and traffic overhead, the first one being the price a user has to pay for anonymity, while the second measuring the cost for the network to support it. As an example we have selected the download of the main page of CNN with Firefox, but other web-pages, the use of other protocols (e.g., ftp or POP/smtp for e-mail), or the use of different browsers do not change the scenario, yielding similar results.

Our goal is to verify if IPpriv can perform better than Tor, a service which has already about 200,000 users despite its performance bottlenecks, which are normally described as a price which must be paid to have anonymity. We have configured IPsec tunnels with the same ciphers as used in Tor (AES-CBC 128-bit encryption and SHA-1 data authentication) in order to guarantee a fair comparison between the systems and be sure that differences are not due to the different efficiency of the encryption algorithm.

To better understand the impact of the overlay networking and the anonymization ‘price’ we have used two sets of overlay nodes for our tests: a circuit of three nodes in our lab (local testbed, lower part of Fig. 3.4); and a circuit of three nodes in different European universities (Internet testbed, upper left part of Fig. 3.4). In both circuits, the
exit node was the same router in our University, therefore the last unencrypted part of the path to CNN was always the same. In both circuits we had Tor as well as IPpriv installed, thus having four different cases. We use direct, non anonymous download as a reference. These five cases are shown on the right side of performance figures.

Additionally, to evaluate the performance also in an “ideal” scenario, where the Internet does not interfere with the download speed, and there is no need for an EN, we have prepared a local copy of the CNN main page on our own server, having five more cases marked with local copy (we show these on the left side of performance figures). Notice that the on-line CNN page is dynamic and changes continuously, while the local copy was always the same (indeed, it turned out to be a page slightly larger than the average), so that on-line and local results are not directly comparable.

The direct connection results refer to non anonymous download, thus representing the baseline performance of the service. All downloads were repeated 10 times averaging the results; error bars refer to ± the standard deviation.

Fig. 3.5 shows the total amount of traffic generated in each scenario. Privacy protection with onion routing induces significant overhead in bytes: around 50% for both solutions. The price of privacy protection is relatively high, but not excessive, and it can

\[1\] To achieve comparable results, our private Tor network was not connected to the global Tor network thus guaranteeing that no other traffic is using the same node modifying performances.
be applied selectively on sensitive traffic. From the network perspective, further overhead is induced by routing packets over multiple overlay links instead of using shortest path routing.

Fig. 3.6 shows average download times. With no bottlenecks (local testbed - local copy), layer 3 operation (2\textsuperscript{nd} column) is clearly much faster than Tor (3\textsuperscript{rd} column). There is almost no delay that can be ascribed to IPsec-based OR operation, in contrast to Tor where the average download time increases threefold (from 1 s to 3 s). Tor’s performance degradation is rooted in architectural choices and the use of TLS tunneling in each overlay hop, and it is not due to bad implementation. The Internet testbed (shown in the 4\textsuperscript{th} and 5\textsuperscript{th} column) obviously increases RTT and thus download time, since packets are routed around Europe to distribute trust to different countries. Here Tor’s disadvantage grows significantly (24 s vs. 6 s with IPsec, again threefold), having a page download time clearly noticeable by the user and thus annoying.

Looking at downloads directly from CNN through the local testbed (right side of Fig. 3.6), IPpriv increases the download time compared to direct download by roughly 50%. Surprisingly, Tor over the local testbed is slower (14 s) than IPsec over the Internet testbed (10 s), indicating that the performance bottlenecks are to be sought for in Tor itself and not in the Internet. Finally, the performance of Tor with the Internet testbed makes it almost unusable with interactive services.
3.5 Summary

This Chapter introduced the idea of supporting anonymous networking (AN) as an extension of IPsec. The presented IPpriv architecture relies entirely on standard IPsec and NAT features to implement the data-plane, including all required encryption, tunneling, forwarding and anonymization functionalities. As far as the control-plane is concerned, our proof-of-concept implementation implements only parts of the required functionality, however, it can be extended with proprietary signaling techniques similar to the ones used in other AN solutions.

AN has not received much attention by the networking community, being considered, until recently, a problem to be tackled at the application level. By supporting privacy-aware, anonymous communications at the network layer with a proof-of-concept implementation of an anonymous overlay embedded within IPsec, we demonstrate that it is possible to enforce anonymity without relying on proprietary application-level routing techniques.

A performance comparison with Tor showed that on the one hand the overhead (in terms of additional transmitted bytes) of IPsec and application level solutions are almost identical, while on the other hand the performance in terms of download delay is 3–4 times faster with the IPsec solution. This performance result, measured using real services over the Internet, is extremely promising and calls for further research to implement signaling solutions to make deployment easy and ‘plug&play’.
Part II

Peer-to-Peer Streaming Overlays
Chapter 4

On the Optimality of Scheduling Algorithms

4.1 Introduction

Chunk and peer selection strategies (or scheduling) are among the main drivers of live P2P streaming system performance. This chapter presents the formal proof that there exists a distributed scheduling strategy which is able to distribute every chunk to all $N$ peers in exactly $\lceil \log_2(N) \rceil + 1$ steps. Since this is the minimum number of steps needed to distribute a chunk, the proposed strategy is optimal. Such a strategy is implementable and an entire class of deadline-based schedulers realize it. We show that at least one of the deadline-based schedulers is resilient to the reduction of the neighborhood size down to values as small as $\log_2(N)$.

4.2 Problem Statement

We study the problem of scheduling (chunk and peer selection) for the dissemination of information to each peer in non-structured overlay networks. It is well known that the lower bound on the dissemination delay of any piece of information, given that nodes have exactly the bandwidth necessary for the streaming itself, is $\delta_{lb} = (\lceil \log_2(N) \rceil + 1)T$ where $T$ is the transmission time of a single chunk. The bound comes from the fact that each node can transmit the information only after receiving it, and the number of nodes owning the chunk at most doubles every $T$. It is also known [37] that centralized schedulers can distribute every chunk of a stream in exactly $\delta_{lb}$. Also, in [11] it was proved that a logarithmic bound holds for several distributed schedulers if $N \to \infty$ and $M_c \to \infty$ ($M_c$ is the number of chunks). However, when real distribution systems are considered, such an asymptotic bound is not equivalent to $\delta_{lb}$.

This chapter focuses on formally proving the existence of a distributed optimal algorithm, and in finding robust, feasible schedulers that with restricted neighborhoods perform within a reasonable bound of the optimal one. These results can form the
starting point (a reference optimum) for further research on heterogeneous systems, on the interaction of the overlay with the underlying IP network, and on all those ‘impairments’ that forbid finding closed-form formal solutions to problems in real networking scenarios.

4.2.1 System Description

We consider an overlay of peers connected with a general mesh topology. The total number of peers is \(N\). Each peer is connected to \(N_N\) other peers, which constitute its neighborhood. A special case is \(N_N = N - 1\), which defines a fully connected mesh. We consider the presence of one more “special peer” that is the source of the video. The source never receives chunks, so its links are logically unidirectional and it is not part of any neighborhood, i.e., its unidirectional links are additional to the others. Fig. 4.1 reports two sample topologies.

The source distributes a live video or TV program. The video is divided into \(M_c\) chunks of equal duration emitted periodically. All peers have unit bandwidth (i.e., they can transmit a chunk in exactly the inter-chunk generation time) on the uplink and no limitations on the downlink. We do not consider churn (i.e., the process of peers joining and leaving the overlay) and we focus, as main performance parameter, on the diffusion delay of chunks, which is the delay with which chunks are received by all peers. Formally, if \(r_i\) is the emission time of chunk \(C_i\), then its diffusion delay is \(f_i = t - r_i\) such that all \(N\) peers have received \(C_i\).

The first scheduling decision is whether a peer pushes information to other peers or if it pulls it from other peers — or a mix of the two policies. Sometimes in the literature it is stated that pushing information is a behavior typical of structured systems, and

\begin{itemize}
  \item \footnote{For the sake of simplicity we restrict discussion to \(n\)-regular topologies: random graphs with symmetric connectivity and \(n\) links per node.}
\end{itemize}
pull methods are more adapted for non-structured overlays. Recent papers like \cite{11,17} use push schedulers on non-structured meshes instead. Indeed, the choice of whether it is better to push or pull information is not related to the structure (or the lack of it) of the system, but to the bandwidth bottleneck, which can create conflicts in scheduling decisions.

Push-based systems are suitable for systems where the bottleneck is the uplink, because this guarantees a priori that only one chunk will be scheduled for transmission on the uplink, and that scheduling conflicts arising from the distributed nature of the scheduling will insist on the downlink of other peers that have more resources and can accept multiple downloads.

Should the situation be reversed (uncommon in networks dominated by ADSL access, but technically possible), then pull-based schedulers would solve a priori the conflict on the downlink, and more bandwidth-endowed uplinks would accommodate scheduling conflicts.

We consider push-based schedulers, but we claim that reversing the bottleneck hypothesis, pull-based schedulers which are dual to those we prove optimal in the sequel can be easily derived.

### 4.2.2 Formal Notation and Definitions

A system is composed of a set \( \mathcal{S} = \{P_1, \ldots, P_N\} \) of \( N \) peers \( P_i \), plus a special node called source. Each peer \( P_i \) receives chunks \( C_j \) from other peers, and sends them out to other peers at a rate \( s(P_i) \). The source sends chunks with rate \( s(\text{source}) \). The set of chunks already received by \( P_i \) at time \( t \) is indicated as \( \mathcal{C}(P_i, t) \).

The source, not included in \( \mathcal{S} \), generates chunks in order, at a fixed rate \( \lambda \) (\( C_j \) is generated by the source at time \( r_j = \frac{1}{\lambda} j \)). We normalize the system w.r.t. \( \lambda \), so that \( r_j = j \). Also, we set \( \forall i, s(P_i) = s(\text{source}) = \lambda = 1 \), which is the limit case to sustain streaming.

If \( \mathcal{D}_j(t - r_j) \) is the set of nodes owning chunk \( C_j \) at time \( t \), the worst case diffusion delay \( f_j \) of chunk \( C_j \) is defined as the time needed by \( C_j \) to be distributed to every peer: \( f_j = \min\{\delta : \mathcal{D}_j(\delta) = \mathcal{S}\} \). According to this definition, a generic peer \( P_i \) will receive chunk \( C_j \) at time \( t \) with \( r_j + 1 \leq t \leq r_j + f_j \). Considering an unstructured overlay \( t \) will be randomly distributed inside such interval. Hence, in an unstructured system \( P_i \) is guaranteed to receive \( C_j \) at most at time \( r_j + f_j \). To correctly reproduce the whole media stream, a peer must buffer chunks for a time of at least \( F = \max_{1 \leq j \leq M}(f_j) \) before starting to play. For this reason, the worst case diffusion delay \( F \) is a fundamental
Chapter 4. On the Optimality of Scheduling Algorithms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>The set of all the peers</td>
</tr>
<tr>
<td>$N$</td>
<td>The number of peers in the system</td>
</tr>
<tr>
<td>$M_c$</td>
<td>The total number of chunks</td>
</tr>
<tr>
<td>$P_i$</td>
<td>The $i^{th}$ peer</td>
</tr>
<tr>
<td>$C_h$</td>
<td>The $h^{th}$ chunk</td>
</tr>
<tr>
<td>$r_h$</td>
<td>The time when the source generates $C_h$</td>
</tr>
<tr>
<td>$N_N$</td>
<td>The neighbourhood size</td>
</tr>
<tr>
<td>$f_h$</td>
<td>The diffusion delay of chunk $C_h$ (the time needed by $C_h$ to reach all the peers)</td>
</tr>
<tr>
<td>$F$</td>
<td>Worst-case diffusion delay ($\max_{1 \leq h \leq M_c}(f_h)$)</td>
</tr>
<tr>
<td>$C(P_i, t)$</td>
<td>The set of chunks owned by peer $P_i$ at time $t$</td>
</tr>
<tr>
<td>$C'(P_i, t)$</td>
<td>The set of chunks owned by $P_i$ at time $t$ which are needed by some of $P_i$’s neighbours</td>
</tr>
<tr>
<td>$N_i$</td>
<td>The neighborhood of peer $P_i$</td>
</tr>
<tr>
<td>$s(P_i)$</td>
<td>The upload bandwidth of peer $P_i$</td>
</tr>
</tbody>
</table>

Table 4.1: Definitions and symbols

performance metric for P2P streaming systems.

At time $t$ the source sends a chunk $C_j$ (with $r_j = t$) to a peer and every peer $P_i$ sends a chunk $C_h \in C(P_i, t)$ to a peer $P_k$. All these chunks will be received at time $t + 1$.

As discussed earlier, the minimum possible diffusion delay $f_j$ for chunk $C_j$ is $\lceil \log_2(N) \rceil + 1$. Chunk diffusion is said to be optimal if $\forall j, f_j = \lceil \log_2(N) \rceil + 1 = F$.

The most important symbols used in this part of the thesis are recalled in Table 4.1.

4.2.3 On Quantiles and Other Statistics

As mentioned previously, our primary performance metric is the worst case diffusion delay $F$. We should note however that besides $F$, quantiles (such as the 90-th, 95-th or 99-th percentile) or other statistics (such as the mean, variance or jitter) of the chunk diffusion delay distribution could also be used to measure performance. We deem that for theoretical analysis, $F$ is the best metric to use for several reasons:

- the human eye is very susceptible to some kinds of losses in video streams, while others might pass unnoticed;
4.2. Problem Statement

- quantifying the effect of losses is difficult and largely depends on encoding decisions such as the codec used, the selected bitrate, or any of the numerous other encoding parameters;

- when \( F \) is finite, a deterministic delay bound can be established — the same cannot be said for quantiles;

- optimality of \( F \) is stronger than the optimality of quantiles. More precisely, the optimality of a quantile follows from the optimality of \( F \), while the reverse is not true.

Quantiles will be used in more experimental sections, in cases where e.g. \( F \) cannot be determined due to some impairments to the system. In Chapters 6 and 7 we will use another approach, getting around the problem of quantiles by deriving a methodology to directly quantify the quality of the received video in case of losses.

Another option would be the characterization of delay with its average, variance, or some kind of jitter interpreted at the chunk level (see [19] for an explanation of various possible interpretations at the packet level). We deem \( F \) and/or percentiles to be a better characterization of the delay distribution in the case of P2P streaming than these statistics.

First of all, average delay is hard to interpret in case of losses. A correct characterization of delay would require at least 2 parameters, making interpretation and comparison cumbersome.

Second, jitter, which is used in some domains such as VoIP, is misleading in our case. In the case of VoIP, higher jitter is considered bad for two reasons: i) it could cause late arrivals, and ii) it increases the size of the required buffer due to possible early arrivals. In the case of P2P streaming, out-of-order delivery and buffering is part of the multi-hop diffusion mechanism, buffer size is usually not regarded as a strict resource constraint. As we will see later, early arrivals of some chunks to some peers are actually the core of efficient re-distribution. Therefore, the main concern is late arrivals, which are much better characterized by \( F \) or quantiles.

4.2.4 Scheduling Peers and Chunks

In a push-based P2P system, when a peer \( P_i \) sends a chunk, it is responsible for selecting the chunk to be sent and the destination peer. The chunk \( C_j \) to be sent is selected by a chunk scheduler, and the destination peer \( P_k \) is selected by a peer scheduler. We focus on algorithms which first select the chunk \( C_j \), and then select a target peer \( P_k \) which
Some well known chunk scheduling algorithms are Latest Blind Chunk, Latest Useful Chunk, and Random Chunk (again, blind or useful).

The Latest Blind Chunk algorithm schedules at time $t$ the latest chunk: $C_j \in \mathcal{C}(P_i, t) : \forall C_h \in \mathcal{C}(P_i, t), r_j \geq r_h$ ($C_j$ is scheduled even if all the other peers already have it).

The Latest Useful Chunk (LUc) algorithm selects a chunk that is needed by at least one peer: $C_j \in \mathcal{C}'(P_i, t) : \forall C_h \in \mathcal{C}'(P_i, t), r_j \geq r_h$ where $\mathcal{C}'(P_i, t)$ is a subset of $\mathcal{C}(P_i, t)$ containing only chunks that have not yet been received (or are not currently being received) by some other peers.

The Random Chunk algorithm selects a random chunk in $\mathcal{C}(P_i, t)$ (Random Blind Chunk) or in $\mathcal{C}'(P_i, t)$ (Random Useful Chunk – RUc).

Once the chunk $C_j$ to be sent has been selected, the peer scheduling algorithms selects a peer $P_k$ which needs $C_j$. The most commonly used peer scheduling algorithm is Random Useful Peer, which randomly selects a peer which needs $C_j$. In theory, the chunk scheduling algorithm can select any $P_k \in S$, but in practice peer $P_i$ will only know a subset of all the other peers, and will select $P_k$ from a subset of $S$ called neighborhood. The neighborhood of $P_i$ is indicated as $\mathcal{N}_i$. The case in which $\forall i, \mathcal{N}_i = S - P_i$ is special, and corresponds to a fully connected graph.

### 4.3 Related Work and Contributions

Optimality of schedulers has been extensively studied in the literature. For the case of full mesh overlay and unit upload bandwidth limits, the generic (i.e., valid for any scheduler) lower bound of $\lceil \log_2(N) \rceil + 1$ is well known. [37] proves that this bound is strict in a streaming scenario by showing the existence of a centralized scheduler that achieves such bound. A similar proof (although for the case of file dissemination) can be found in [45]. Our work improves on these results by proving the existence of distributed schedulers that achieve the same strict bound.

Generic upper bounds as well as upper bounds on the distribution times achieved by different distributed schedulers can also be found in literature. The fundamental work of [50] studies asymptotic properties of distributed gossiping algorithms in a similar setting, showing an upper bound for any pull based distribution of all the messages in $O(M \log(N))$ time with high probability even for blind algorithms. Generic asymptotic bounds are also shown for blind push based algorithms, although in this case full
dissemination cannot be guaranteed. A blind algorithm that distributes chunks with a high probability in \((9 \times \text{M}_c + 9 \times \log_2(\text{N}))T\) is also shown. Note that this suggests a distribution delay for the individual chunk that grows with \(\text{M}_c\).

The authors of [45] also evaluate blind distributed strategies in the case of file distribution, showing distribution delays dependent on the number of chunks. [11] studies upper bounds for specific well known algorithms, showing that the combination of random peer selection and latest useful peer selection achieves asymptotically good delays, however this demonstration is provided in the case of upload bandwidth higher than 1.

The distributed scheduler presented in this chapter performs significantly better than the generic upper bounds shown in [50] and [45] in that it achieves full diffusion of all chunks in \((\text{M}_c + \lceil \log_2(\text{N}) \rceil)T\), i.e. a chunk diffusion delay independent of \(\text{M}_c\).

It also differs from the streaming algorithms studied in [11], since this strict delay bound holds for any \(\text{N}\) (not just asymptotically), and it is valid even in the boundary case of unit upload bandwidth, without relying on redundant source coding.

[11] uses Erdős-Rényi (ER) random graphs to model the restricted neighborhood. With \(\text{N} = 600\) and \(\overline{\text{N}} = 10\), the authors find that the studied algorithms suffer significant losses. These chunk losses are confirmed by our results (even if our random graph model is slightly different) for the algorithms considered therein. However, we also show (through simulations) that the new algorithm performs near the optimum with any \(\text{M}_c\) and any \(\text{N}\), even with significant overlay restrictions. Namely, reducing the neighborhood size to any \(\overline{\text{N}} \geq \lceil \log_2(\text{N}) \rceil\), our algorithm keeps distributing all chunks with a delay only slightly above the lower bound and always (on all simulated \(\overline{\text{N}}\)-regular random graphs) below \(2 \times (\lceil \log_2(\text{N}) \rceil + 1)T\). Note that the neighborhood of \(\lceil \log_2(\text{N}) \rceil\) practically means less than 30 in any reasonable setting.

### 4.4 Optimal Peer Scheduling

Schedulers can be analyzed from several perspectives. A key observation when looking for a scheduler that provides strict optimality can be derived by looking at schedulers from the redistribution perspective: to achieve optimal diffusion delay, it must be ensured that the number of peers owning a given chunk doubles in each cycle. In other words, a peer that receives a chunk should immediately start redistributing it, and it should continue the distribution of the same chunk till it gets diffused in the whole system.

The problem with e.g. random useful peer selection is that the selected peer might be busy redistributing another chunk. Thus, when receiving the new chunk, it should
either interrupt the diffusion of the previously distributed chunk, or it will not be able to start redistributing the new chunk.

This observation leads us to the definition of the “Earliest-Latest” peer scheduler (ELp) as follows: when executed in peer \( P_p \), ELp selects as target a peer \( P_l \) that needs \( C_h \) and owns the latest chunk \( C_k \) with the earliest generation time \( r_k \):

\[
C_h \not\in C(P_l, t) \land \forall P_j \in N_p, L(P_l, t) \leq L(P_j, t)
\]  \hspace{1cm} (4.1)

where \( L(P_i, t) = \max_k \{r_k : C_k \in C(P_i, t)\} \) is the latest chunk owned by or in arrival to \( P_i \) at time \( t \). If at time \( t \) \( P \) has not received any chunk yet, \( L(P_i, t) = 0 \). If more peers exist that satisfy (4.1) one is chosen at random.

4.4.1 Analysis of ELp

In this section, some important properties of the LUc/ELp scheduling algorithms are proved for the case of a fully connected overlay. In Theorems 1 and 2 it is proved that LUc/ELp achieves optimality.

**Lemma 1.** When using ELp, \( \forall i, t \leq \lfloor \log_2(N + 1) \rfloor \Rightarrow ||C(P_i, t)|| \leq 1 \).

*Proof.* During an initial transient, at time \( t \) the system contains \( 2^t - 1 \) chunk instances (because at every time instant the source emits a new chunk and all the peers having at least one chunk send a chunk); hence, there are \( N - (2^t - 1) \) peers having no chunks. By definition, the ELp scheduler selects such peers as targets, hence a peer \( P_i \) can have more than 1 chunk only if \( 2^t - 1 > N \Rightarrow 2^t > N + 1 \Rightarrow t > \log_2(N + 1) \). \( \Box \)

**Lemma 2.** If \( \forall i, s(P_i) = \lambda = 1 \land N_i = S - P_i \), if a LUc/ELp scheduling algorithm is used, then

\[
\forall \delta, \ 0 < \delta \leq \lfloor \log_2(N) \rfloor \Rightarrow ||L_j(\delta)|| = 2^{\delta - 1}
\]

where \( L_j(\delta) = \{P_i : \max_k \{r_k : C_k \in C(P_i, r_j + \delta)\} = r_j\} \) is the set of peers having \( C_j \) as their latest chunk at time \( r_j + \delta \).

*Proof.* The lemma is proved by induction on \( \delta = t - r_j \), and by considering the latest chunk owned by the peers at time \( t = r_j + \delta \), so that \( S \) is partitioned into three subsets:

- \( \mathcal{X}(\delta) = \bigcup \{L_j(i) : i > \delta\} \) is the set of peers with latest chunk later than \( C_j \);
- \( \mathcal{Y}(\delta) = L_j(\delta) \) is the set of peers having \( C_j \) as their latest chunk;
- \( \mathcal{Z}(\delta) = \bigcup \{L_j(i) : i < \delta\} \) the set of peers with latest chunk earlier than \( C_j \).
The above is a partitioning into disjoint subsets, therefore \(|\mathcal{X}(\delta)| + |\mathcal{Y}(\delta)| + |\mathcal{Z}(\delta)| = |\mathcal{S}| = N\). The lemma can be now proved by induction on \(\delta\).

**Induction base:** After chunk \(C_j\) is generated by the source at time \(r_j\), it is sent out to a peer \(P_i\), which will receive it at time \(t = r_j + 1 \Rightarrow \delta = 1\). Hence,

\[
\mathcal{D}_j(1) = \{P_i\} \Rightarrow |\mathcal{D}_j(1)| = 1
\]

As \(C_j\) is the newest chunk in the system, \(\mathcal{X}(\delta)\) is empty and \(C_j\) becomes the latest chunk on \(P_i\):

\[
\forall C_k \in \mathcal{C}(P_i,r_j+1), r_j > r_k
\]

Thus, \(\delta = 1 \Rightarrow |\mathcal{L}_j(\delta)| = |\mathcal{D}_j(\delta)| = 1 = 2^{\delta-1}, |\mathcal{X}(\delta)| = 0 = 2^{\delta-1} - 1\). Also note that \(|\mathcal{Z}(\delta)| = N - 1 > |\mathcal{X}(\delta)| + |\mathcal{Y}(\delta)|\).

**Inductive step:** First of all, it is easy to notice that \(|\mathcal{X}(\delta - 1)| \leq 2^{\delta-2} - 1\): in fact, at every time unit a new chunk \(C_k : r_k > r_j\) is generated, and all the peers \(P_i \in \mathcal{X}(k - 1)\) can send their latest chunk to another peer. As a result, \(|\mathcal{X}(\delta - 1)|\) will be at most equal to \(2|\mathcal{X}(\delta - 2)| + 1\). But \(|\mathcal{X}(\delta - 2)| \leq 2^{\delta-3} - 1\) (by induction), so

\[
|\mathcal{X}(\delta - 1)| \leq 2(2^{\delta-3} - 1) + 1 = 2^{\delta-2} - 1
\]

Now, if \(\delta \leq \lfloor \log_2(N) \rfloor\), then

\[
\delta \leq \lfloor \log_2(N) \rfloor \Rightarrow 2^\delta \leq N \Rightarrow 2(2^{\delta-2} + 2^{\delta-2}) \leq N
\]

and since \(|\mathcal{L}_j(\delta - 1)| = 2^{\delta-2}, |\mathcal{X}(\delta - 1)| \leq 2^{\delta-2} - 1\) and \(|\mathcal{Z}(\delta - 1)| = N - |\mathcal{X}(\delta - 1)| - |\mathcal{Y}(\delta - 1)|\), the above equation can be rewritten as

\[
2(|\mathcal{X}(\delta - 1)| + 1 + |\mathcal{Y}(\delta - 1)|) \leq N \Rightarrow |\mathcal{X}(\delta - 1)| + |\mathcal{Y}(\delta - 1)| \leq |\mathcal{Z}(\delta - 1)| - 2
\]

As a result, at \(\delta - 1\), \(|\mathcal{Z}(\delta - 1)|\) is more than half of \(N\), therefore there are enough peers with latest chunk older than \(C_j\) to receive chunks from both \(\mathcal{X}(\delta - 1)\) and \(\mathcal{Y}(\delta - 1)\), so \(|\mathcal{L}_j(\delta)| = |\mathcal{D}_j(\delta)| = 2^{\delta-1}\), hence the claim.

**Theorem 1.** If \(N = 2^i\), algorithm \(LUc/ELp\) is optimal.

**Proof.** By definition, an algorithm is optimal iff \(\forall j, f_j = \lfloor \log_2(N) \rfloor + 1\). In this case, this means \(\forall j, f_j = i + 1\). By Lemma 2

\[
\forall j, \delta \leq \lfloor \log_2(N) \rfloor \Rightarrow |\mathcal{L}_j(\delta)| = 2^{\delta-1}
\]

hence, \(\forall j, |\mathcal{L}_j(i)| = 2^{i-1}\). As a result, \(|\mathcal{D}_j(i + 1)| = 2|\mathcal{L}_j(i)| = 2^i = N\), and \(f_j = i + 1\). \(\square\)
Theorem 2. Algorithm LUc/ELp is optimal also if $N \neq 2^i$. 

Proof. If $N = 2^i + n$, with $n < 2^i$, by Lemma 2 it comes $\forall j, ||L_j(i)|| = 2^{i-1}$. Hence, for $\delta = i$ chunk $C_j$ is sent $2^{i-1}$ times and chunks with $r_k > r_j$ are sent $2^{i-1} - 1$ times. As a result, $||D_j(i+1)|| = 2^i, ||X(i+1)|| = 2^i - 1, ||Z(i+1)|| = 0$, and $||L_j(i+1)|| < ||D_j(i+1)||$. To compute the exact value of $||L_j(i+1)||$, let $x$ be the number of chunks sent by peers in $X(i)$ to peers in $Z(i)$ and let $y$ be the number of chunks sent by peers in $Y(i)$ to peers in $Z(i)$. According to the peer scheduling rules, $x + y = ||Z(i)||$ (because chunks are sent to peers having the earliest latest chunk). Moreover, $||L_j(i+1)|| = y + ||L_j(i)|| - (||X(i)|| - x)$. Hence,

$$||L_j(i+1)|| = ||Z(i)|| - x + 2^{i-1} - (2^{i-1} - 1 - x) =$$

$$(N - 2^{i-1} - (2^{i-1} - 1)) - x + 2^{i-1} - 2^{i-1} + 1 + x = N - 2^i + 1 + 1 = N - 2^i + 2$$

Finally,

$$||D_j(i+2)|| = \min\{N, ||D_j(i+1)|| + ||L_j(i+1)|| = 2^i + N - 2^i + 2\} = N$$

Hence, $f_j = i + 2 = \lceil\log_2(N)\rceil + 1$. 

Observation 1. If an optimal chunk scheduling is used, all the copies of every chunk $C_k$ are forwarded from time $r_k$ to time $r_k + f_k - 2$.

4.5 Optimal Chunk Scheduling

We have shown in Theorems 1 and 2 that a LUc/ELp scheduler is optimal in the full mesh case; however, LUc/ELp provides large worst-case diffusion delays when the neighbourhood size is reduced (as will be shown in Section 4.6). Such a bad behaviour is common to all the LUc schedulers, and is caused by the fact that such schedulers always select the latest useful chunk. Hence, if for some reason (such as a restricted neighbourhood size or a limited knowledge of the neighbourhood) a chunk $C_k$ with $r_k > r_h$ arrives to a peer before $C_h$ is completely diffused, then the peer is not able to diffuse $C_h$ anymore and its diffusion delay is increased by a large amount. In other words, every time that limited knowledge of the neighborhood makes a later chunk arrive to a peer before an earlier one, the diffusion of this latter might be stopped.

For this reason, a new chunk scheduling algorithm has been developed to be equivalent to LUc/ELp in the full mesh case, and to perform reasonably well when the graph is not fully connected. The new algorithm is based on a deadline-based chunk scheduling algorithm, named DLc. The DLc scheduling algorithm works based on scheduling
4.5. Optimal Chunk Scheduling

deadlines $d_k$ associated to every chunk instance. The scheduling deadline is initialized to $d_k = r_k + 2$ when the source sends $C_k$ at time $r_k$. The chunk scheduler then works by selecting the chunk $C_k$ with the minimum scheduling deadline:

$$C_k : \forall C_h \in C'(P_i, t), d_k \leq d_h; \quad (4.2)$$

Before sending $C_k$ its scheduling deadline is postponed by 2 time units: $d_k = d_k + 2$ (both $P_i$ and the destination peer will see $C_k$ with its new scheduling deadline, while chunk instances present in other peers are obviously not affected).

The scheduling strategy based on selecting the chunk with a minimum deadline is known in literature as Earliest Deadline First (EDF), and is mentioned as “Deadline Driven Scheduling” in a seminal paper by Liu and Layland [36], but to the best of our knowledge, it has never been applied with dynamic deadlines in distributed systems.

**Observation 2.** The scheduling deadline $d_k$ of a chunk instance $C_k$ at peer $P_i$ is equal to $r_k + 2d$, where $d$ is the number of times that $C_k$ has been selected by the DLc schedulers along the path taken by the chunk till $P_i$. $d$ is larger or equal to the length of the path, since $d_k$ is increased both when $C_k$ was selected and sent along the path, and also when it was selected but sent to other peers.

4.5.1 Analysis of DLc with Full Meshes

Based on the optimality of LUc/ELp, it is possible to prove that DLc/ELp is an optimal algorithm too. This is done by showing that on a full mesh it generates the same schedule as LUc/ELp.

**Theorem 3.** If $\forall i, s(P_i) = \lambda = 1, \forall i, N_i = S - P_i$, then the chunk distribution produced by DLc/ELp is identical to the chunk distribution produced by LUc/ELp.

**Proof.** By contradiction: assume that at any time $t_0$ the chunk distribution produced by DLc/ELp starts to differ from the one produced by LUc/ELp, i.e., assume that DLc at peer $P_i$ at time $t_0$ selects chunk $C_j$ while LUc would select chunk $C_k$ (so, $r_k > r_j$). However, it will be shown that choosing $C_j$ with DLc implies $r_j \geq r_k$ contradicting the hypothesis $r_k > r_j$.

If $t_0 < \lfloor \log_2(N+1) \rfloor$, then Lemma guarantees that all chunk schedulers are identical under ELp peer scheduling.

If $t_0 \geq \lfloor \log_2(N + 1) \rfloor$, we have from the hypotheses that $\forall t < t_0$ the schedules produced by DLc/ELp and LUc/ELp are identical. By definition at time $t_0$ in $P_i$ LUc/ELp chooses

$$C_k \in C'(P_i, t_0) : \forall C_h \in C'(P_i, t_0) r_k \geq r_h.$$
Since the source only produces a single chunk at every time unit, \( r_k \) and \( r_h \) cannot have the same value, hence \( r_k > r_h \). To obtain a different schedule DLc/ELp must choose \( C_j \neq C_k \).

Since for \( t < t_0 \) DLc/ELp produced the same schedule as LUc/ELp, \( C_j \) and \( C_k \) have been transmitted \( t_0 - r_j \) and \( t_0 - r_k \) times respectively (see Observation 1); hence, \( d_i = r_i + 2(t_0 - r_i) \) for both \( C_j \) and \( C_k \).

Since DLc/ELp chooses \( C_j \in C'(P_i, t_0) : \forall C_h \in C'(P_i, t_0) \) \( d_j \leq d_h \) we have \( d_j \leq d_k \Rightarrow r_j + 2(t_0 - r_j) \leq r_k + 2(t_0 - r_k) \Rightarrow -r_j \leq -r_k \Rightarrow r_j \geq r_k \) which contradicts the hypothesis \( r_k > r_j \).

**Observation 3.** Note that the DLc scheduler postpones the scheduling deadline by two time units per transmission as \( d_k = d_k + 2 \). If a generic constant \( q \) was used instead of 2 and the scheduling deadline was postponed as \( d_k = d_k + q \), then the last equation in the proof of Theorem 1 would have become

\[
    r_j + q(t_0 - r_j) \leq r_k + q(t_0 - r_k) \Rightarrow (q - 1)r_j \geq (q - 1)r_k
\]

which contradicts \( r_k > r_j \) if \( q > 1 \). Hence, if a generic constant \( q > 1 \) is used to postpone the scheduling deadline, then DLc/ELp is still equivalent to LUc/ELp. In this sense, DLc can be seen as a whole class of deadline-based algorithms.

### 4.6 Neighborhood Restriction and Selected Results

Although both LUc/ELp and DLc/ELp have been proved to provide optimal performance in the case of a fully connected graph, their performance in more realistic situations is still unclear. Besides these two algorithms we consider various combinations with LUc, RUc and RUp algorithms for comparison.

#### 4.6.1 Simulating P2P Streaming and Measuring Performance

The behavior of the scheduling algorithms introduced in Section 4.4 is analyzed using the SSSim simulator [C12], which has been specifically developed to perform optimality tests on new scheduling algorithms. For this purpose, SSSim has been developed as a discrete-time simulator (opposed to an event-driven simulator, that would be less efficient in this case, requiring an event queue and all the related overhead). However, the simulator has been designed to be modular and flexible (when this does not affect performance), so most of its modules can be re-used in event-driven simulations as well. The simulator is organised in software modules, interfaced using a clean and well defined API: the overlay
4.6. Neighborhood Restriction and Selected Results

Scheduling performance is analyzed by setting up an overlay of $N$ peers with unit upload and infinite download bandwidth. The source distributes $M_c$ chunks. As explained in Section 4.2 the performance metric considered is the worst case diffusion time $F$, and a scheduling algorithm is optimal iff $F = \lceil \log_2(N) \rceil + 1$.

![Figure 4.2: Full mesh overlay; maximum diffusion delay as a function of N; 500 chunks](image)

First of all, the algorithms have been simulated on a fully connected graph, as shown in Figure 4.2. In accordance with Theorems 2 and 3, LUc/ELp and DLc/ELp achieve optimal performance, outperforming the other algorithms (in particular, RUc/ELp achieves a value of $f_i$ near the double of the optimal, and all the other algorithms achieved even worse performance).

4.6.2 Restricting the Overlay

In realistic situations a restricted overlay is used instead of a fully connected graph. Such a restricted overlay is modeled assuming bidirectional relations and a pre-defined number ($N_N = ||N_i||$) of neighbor nodes. The resulting graph is a random $N_N$-regular graph. In the following simulations, the algorithms are evaluated on 10 instances of the random $N_N$-regular graph.

The left hand side of Figure 4.3 shows performance of different streaming algorithms as a function of $N_N$ and shows how the LUc/ELp algorithm (which is optimal on a full mesh) is highly sensitive to neighborhood restrictions and performs badly when $N_N < N - 1$. DLc/ELp, on the other hand, works better than all the other algorithms and is able to achieve values of $F$ near the optimum (which in this case is 11). The right side of Figure 4.3 shows how the number of chunks affects $F$ for $N_N = 11$ (note that $\log_2(N) = 9.9658$). DLc/ELp keeps good performance even for long streams, while for
several other algorithms $f_i$ increases with $i$ (the performance of the algorithm depends on the stream length), hence these distribution mechanisms result to be unstable in a streaming context.

### 4.6.3 Limiting the Chunk Buffer Size

The only solution to the instability problem is to define a playout delay $D$, and to discard chunks $C_j$ at time $r_j + D$. This causes some chunk loss (for chunks $C_j$ that would have $f_i > D$), but can make the distribution system stable again. Moreover, the playout delay $D$ can be used to dimension the chunk buffers in the peers (in particular, each peer needs to buffer at most $D$ chunks$^2$).

Since some chunks can be lost, the performance should be evaluated based on both chunk loss ratio and the maximum delay. Figure 4.4 plots the chunk loss ratio (left) for the various algorithms as a function of the neighborhood size with $D = 32$. Note that for $N_N > 14$ the chunk loss ratio for DLc/ELp is 0, showing that it is possible to dimension the chunk buffer size so that it does not affect the algorithm’s performance (to the authors’ best knowledge, this is not possible for the other algorithms). The worst case diffusion time $F$ (right) fastly approaches the optimum with DLc/ELp, while it is obviously 32 for all other algorithms.

---

$^2$Implementing the chunks buffer size in the simulator can enable optimizations which allow to simulate larger task sets, hence we move to $N = 10000$. 

---
Figure 4.4: Chunk loss and F as a function of the neighborhood size ($N = 10000, D = 32$)

4.7 Summary

This chapter presented the formal proof that distributed algorithms can achieve optimal diffusion for streaming applications in unstructured meshes. The chapter introduced a class of deadline based algorithms DLc/ELp which are optimal in full meshes and maintain very good properties also in realistic scenarios with small neighborhoods.
Chapter 5

Bandwidth-Aware Scheduling Algorithms

5.1 Introduction

Contrary to the assumptions of the model used in Chapter 4, networks are neither ideal nor homogeneous: peers’ upload bandwidth shows large variation due to differences in access technologies and network conditions. Therefore, the choice of the destination peer affects how the specific chunk will be diffused in the future. Peers more endowed with resources will diffuse the chunk more efficiently than peers with scarce resources.

Heterogeneous scenarios have already been considered in the literature [11, 17, 37, 39], but only few of such works (namely, [17, 37]) explicitly used bandwidth information in peer scheduling. Moreover, most of the previous works considered few possible bandwidth scenarios only, whereas in this chapter, three different kinds of scenarios (3-class, uniform, and free-riders) are studied and the proposed algorithm appears to perform better than the other algorithms in all the considered cases.

5.2 A Bandwidth-Aware ELp Algorithm

A peer scheduler, in order to be robust to different bandwidth distribution scenarios, should be both bandwidth aware [17] and select peers that are in the best conditions to redistribute the chunk as ELp does.

A first way to integrate ELp with some kind of bandwidth awareness is to use hierarchical scheduling. Two possible hierarchical combinations can be implemented: ELBA and BAELp. ELBA uses ELp to select a set of peers having the earliest latest chunk, and then uses a bandwidth aware strategy to select the peer having the highest output bandwidth among them. BAELp first uses a bandwidth aware scheduler to select the set of peers having the highest output bandwidths and then applies ELp scheduling to this set.

Although hierarchical scheduling can be very effective in some situations, a better
function \( L(P_k, t) \)
\[
\text{max} = 0 \\
\text{for all } C_h \text{ in } C(P_k, t) \text{ do} \\
\quad \text{if } r_h > \text{max} \text{ then} \\
\quad \quad \text{max} = r_h \\
\quad \text{end if} \\
\text{end for} \\
\text{return max}
\]
end function

function \( \text{ELP}(P_i, t) \)
\[
\text{int min; peer target;} \\
\text{for all } P_k \text{ in } N_i \text{ do} \\
\quad \text{if } L(P_k, t) < \text{min} \text{ then} \\
\quad \quad \text{min} = L(P_k, t); \\
\quad \quad \text{target} = P_k; \\
\quad \text{end if} \\
\text{end for} \\
\text{return target}
\]
end function

Figure 5.1: The ELp algorithm.

integration of the two scheduling algorithms can improve the system’s performance (as it will be shown in Sect.5.3). To understand how to integrate ELp and bandwidth aware scheduling, for the convenience of the reader, we repeat the algorithmic definition of ELp from Chapter 4 in Fig. 5.1.

Let \( L(P_i, t) \) be the latest chunk owned by \( P_i \) at time \( t \). ELp minimises \( L(P_j, t) \) in \( N_i \), which is equivalent to maximising \( t - L(P_j, t) \). The meaning of this rule is that ELp tries to select the target peer \( P_i \) having the latest chunk that has been distributed within the overlay for more time (hence, it can be argued that such a chunk will be useful for less time, and the peer can soon pass to distribute another one).

When the output bandwidths \( s(P_j) \) of the various peers are not uniform, maximising \( t - L(P_j, t) \) is not meaningful. On the one hand, when \( P_i \) sends a chunk to \( P_j \) at time \( t \), this chunk will be received at time \( t' = t + 1/s(P_i) \) (measuring bandwidths in chunks per second). On the other hand, a peer \( P_j \) with bandwidth \( s(P_j) \) might redistribute chunks
faster or slower than other neighbours of $P_i$. This suggests to modify the ELp peer scheduler to select target peers based on the following multi-metric selection strategy at any given peer $P_i$:

$$P_j \leftarrow \max_k \{ (t - L(P_k, t)) + \beta \frac{s(P_k)}{s(P_i)} \}$$  \hspace{1cm} (5.1)

and $\beta$ is an apportioning coefficient to blend the relative importance of the “earliest-latest” principle with the bandwidth awareness. The use of the ratio $[s(P_k)/s(P_i)]$ instead of the absolute bandwidth is simply a normalization to avoid that different stream rates (e.g., HDTV instead of standard definition) require the selection of different apportioning coefficients: (5.1) is normalized w.r.t. both the chunk emission time $T_s$ and the stream bitrate.

Note that the weight $\beta$ enables to realise trade-offs between $BA_{ELP}$ and $EL_{BAP}$: for $\beta >> 1$ maximizing (5.1) tends to $BA_{ELP}$, whereas for $\beta << 1$ the algorithm tends to $EL_{BAP}$. Finally, note that if all the peers have the same output bandwidth then maximizing (5.1) is equivalent to ELp regardless of $\beta$. The peer scheduling algorithm resulting from this heuristic is named $BA_{ELP}$ (Bandwidth Aware Earliest Latest peer scheduler).

### 5.3 Network and Bandwidth Models

We restrict the analysis to the case when bandwidths are a property of the single peer and the overlay topology is n-regular or a full mesh. The bandwidth being a property of the peer is typical of cases when the bottleneck is on the access link as in ADSL. We assume perfect knowledge of the neighbourhood status, which corresponds to situations when the delays are small compared to chunk transmission times. The download bandwidth of a peer $P_i$ is assumed to be much larger than its upload bandwidth $s(P_i)$.

Since the upload bandwidths of peers $P_i \in S$ are not homogeneous, it is important to model their distribution. We consider two possible types of bandwidth distributions: *class-based distributions* (such as may arise from ADSL links), and *continuous distributions* (that may arise from user-imposed constraints). In a class-based distribution, the peers are grouped in a finite number of classes, and each class of peers is characterised by a different upload bandwidth. In a continuous distribution, the upload bandwidth of every single peer is randomly assigned according to a specified Probability Density Function (PDF).

---

1 In n-regular topologies all nodes have the same connectivity degree, which means that the neighbourhood size $N_N$ is equal for all peers.
Various models of class-based and continuous bandwidth distributions have been tested and simulated. We consider results generated in three different scenarios referred to as “3-class”, “uniform”, and “free-riders”. The 3-class scenario is a class-based distribution with three classes: low-bandwidth peers (having an upload bandwidth equal to $0.5\bar{B}$), mid-bandwidth peers (having an upload bandwidth equal to $\bar{B}$), and high-bandwidth peers having an upload bandwidth equal to $2\bar{B}$. The fraction of high-bandwidth peers in the system is $h/3$ (where $h$ is the heterogeneity factor of this scenario), the fraction of low-bandwidth peers is $2h/3$, and the fraction of mid-bandwidth peers is $1 - h$; as a result, the average bandwidth is

$$2\bar{B} \cdot h/3 + \bar{B} \cdot (1 - h) + 0.5\bar{B} \cdot 2h/3 = \bar{B}$$

(5.2)

This scenario has been selected because it captures the most important properties of a class-based distribution, and a similar setup has already been used in literature [11].

The uniform scenario is an example of continuous distribution, in which $s(P_i)$ is uniformly distributed between a minimum value $B^{\text{min}} = (1 - \Delta B)\bar{B}$ and a maximum value $B^{\text{max}} = (1 + \Delta B)\bar{B}$, where $\bar{B}$ is the average upload bandwidth and $\Delta B$ is the bandwidth variability (a measure of heterogeneity) of this scenario. This scenario is particularly interesting because it allows checking if some of the properties and results noticed in the 3-class scenario are due to artifacts caused by a class-based distribution, or are generic properties.

Finally, the free-riders scenario is based on a two-class distribution in which one class of peers is composed of “free riders”, having upload bandwidth equal to 0. This scenario is important as it allows understanding what happens when some of the peers for some reasons do not contribute to the chunks diffusion. The average bandwidth $\bar{B}$ is kept constant also in this scenario.

### 5.4 Comparison of Algorithms

In what follows we compare the performance of various peer schedulers in three heterogeneous bandwidth scenarios. In addition to the previously introduced RUp, ELp, BAELp, $B_{AELP}$ and $EL_{BAP}$ peer schedulers, we also compare performance to the following peer selection strategies.

---

$^2$Note that free-riding behavior is considered malicious and punished though incentives in some systems, while it is allowed in others. Here we only concentrate on the scheduler’s capability to support free riders when desired, without going into details of discussing the value of incentivization schemes.
5.4. Comparison of Algorithms

- **Most Deprived Peer (MDp):** $P_i$ selects the peer $P_j$ in $\mathcal{N}_i$ which owns the smallest number of the chunks currently owned by $P_i$.

- **Bandwidth Aware scheduler (BAWp):** $P_i$ randomly selects a target peer $P_k \in \mathcal{N}_i$ (as in RUp); the probability of selecting $P_k$ is proportional to its output bitrate $s(P_k)$. 

To simplify comparison and limit the number of performance curves, we only combine these with the DLc chunk scheduler in chunk-first order. DLc and BAELp has two tuning parameters: the deadline postponing $\delta$ of DLc and the bandwidth apportioning $\beta$ of BAELp. Throughout this section we use $\delta = 5$ and $\beta = 3$. See Section 5.5 for a detailed study of the effect of $\delta$ and $\beta$ on scheduling performance. All results in this section are averaged over 100 simulation runs (with different topologies generated at random); the error bars indicate the confidence interval with 95% confidence level.

Fig. 5.2 plots the worst case ($F$) and the 90-th percentile ($F_{90}$) of the diffusion delay for the scheduling algorithms under analysis, as a function of the heterogeneity $h$ in a 3-class scenario. The overlay has $N = 600$ peers connected in full mesh and $M_c = 1200$ chunks are distributed. The average upload bandwidth is 1.

First of all, $F_{90}$ and $F$ behaviour are very similar (for this reason, in the rest of this section only $F_{90}$ will be shown). Second, the peer scheduling algorithms can be classified in 3 groups. The first group, including BAELp, EL_BAP, and ELp is the one showing consistently the best performance regardless of the heterogeneity factor.Schedulers of the second group (MDp, RUp, and BA_Wp) perform identically to each other when the peers are homogeneous ($h = 0$), however BA_Wp handles heterogeneity better than the other two. The third group, finally, which only contains BA_ELP, provides reasonable performance only for the homogeneous system and very heterogeneous ones. This strange behavior is due to the strict bandwidth-based ordering found in this peer scheduler: when heterogeneity is small (but not 0), high bandwidth peers represent only a small subset of the peers, but get all the attention. 

Building and maintaining a full mesh overlay is not realistic as the number of peers increases; moreover, the playout delay is finite and small, limiting the maximum delay a chunk can have before being discarded. As an example of a more realistic scenario, we selected a neighbourhood size equal to 20 and a playout delay of 50 (other numbers would not change the meaning and quality of the results).

Results are shown in Fig. 5.3 which reports $F_{90}$ for the 3-class (upper plot) and uniform (lower plot) scenarios. In both plots BAELp outperforms all other algorithms. BA_ELP shows an $F_{90}$ equal to the playout delay, which means chunks are missed, and
Chapter 5. Bandwidth-Aware Scheduling Algorithms

Figure 5.2: 3-class scenario with full mesh: diffusion delay as a function of the heterogeneity factor $h$; Upper plot: $F_{90}$, Lower plot: $F$; $\overline{B} = 1$, $N = 600$ peers in full mesh, $M_c = 1200$ chunks.

the quality will be poor.

Observing both Figs. 5.2 and 5.3 two remarks should be made. First of all, reducing the neighbourhood size tends to increase the gap between the first and the second group of schedulers. Second, in the uniform scenario, while the order of schedulers is unchanged, the difference decreases. This second result is due to a smaller actual heterogeneity of the uniform scenario w.r.t. the 3-class one.

In the next set of experiments, the stringent (and somewhat arbitrary) $\overline{B} = 1$ assumption has been removed, comparing the behaviour of the schedulers when the average upload bandwidth is increased. Fig. 5.4 reports $F_{90}$ as a function of the average bandwidth $\overline{B}$. The neighbourhood size is 20 and the playout delay is 50. The upper plot refers to the 3-class scenario with $h = 0.5$, and the lower plot to the uniform scenario.
Figure 5.3: $F_{90}$ as a function of heterogeneity for neighbourhood size 20 and playout delay 50; Upper plot: 3-class, Lower plot: Uniform; $\overline{B} = 1$, $N = 600$ peers, $M_c = 2000$ chunks.

with $\Delta_B = 0.8$. Again, BAELp is the best performing algorithm (clearly, when the average bandwidth increases too much, the differences between the various algorithms vanish).

Finally, we explore the ability to cope with peers that do not contribute to the stream distribution. In this “free-riders” scenario the neighbourhood size must be increased, otherwise the system becomes fragile when too many peers do not cooperate: we set the neighbourhood size to 100 peers. Fig. 5.5 reports $F_{90}$ as a function of the fraction of peers that do not redistribute chunks (free riders, having upload bandwidth equal to 0). Note that curves relative to the ELp, RU, MDp, and EL_BAP schedulers stop between 15% and 20% of free riders, because with higher fractions of free riders such schedulers are not able to properly diffuse the stream. Again, this experiment shows
that BAELp is the most resilient scheduler in face of changing conditions. Note that BAELp performance first improves before it starts to degrade due to the number of free-riders. Recall that we have defined the scenario such that the average bandwidth is kept constant. Thus, as the number of free-riders grows, the bandwidth of each peer in the high bandwidth class also increases. The initial performance improvement is most probably due to this bandwidth increase. Later, when the number of free-riders is high, performance is mostly determined by the large number of non-contributing peers.
Figure 5.5: Free Riders scenario, $\overline{B} = 1$, neighbourhood size 100 and playout delay 50: $F_{90}$ versus the fraction of the free riders. $\overline{B} = 1$, $N = 1000$ peers, $M_c = 2000$ chunks.
Chapter 5. Bandwidth-Aware Scheduling Algorithms

5.5 Sensitivity Analysis with respect to $\beta$ and $\delta$

In this section, the effects of the tuning parameters of DLc/BAELp are analysed. DLc/BAELp has two tuning parameters: the deadline postponing $\delta$ of DLc and the bandwidth apportioning $\beta$ of BAELp. In what follows we explicitly indicate their values using the following notation: $DL_{\delta}c/BA_{\beta}ELp$. We restrict our study to the 3-class heterogeneous bandwidth scenario.

Fig. 5.6 presents a sensitivity analysis of $BA_{\beta}ELp$ for a scenario with medium heterogeneity ($h = 0.5$). Chunk schedulers are varied to verify if the behavior of $BA_{\beta}ELp$ is strongly dependent on the associated chunk scheduler or not. We use four settings of $DL_{\delta}c$ with different $\delta$ plus LUc and RUc. The number of peers is 1000, the video is 2000
5.5. Sensitivity Analysis with respect to $\beta$ and $\delta$

Figure 5.7: $DL_{\delta c}$ sensitivity to $\delta$ for different heterogeneity factors $h$: $B = 1$, $N = 1000$, $M_c = 2000$, neighborhood size 20 and playout delay $50T_s$, BA$_3$ELp peer scheduler

chunks long, the neighborhood size is 20, and the playout delay is fixed at $50T_s$. The lower plot reports the maximum delay and the upper one the 95-th percentile. Points were obtained as an average of 100 simulation runs; bars show 95% confidence intervals. The behavior of BA$_3$ELp as a function of $\beta$ is practically independent from the chunk scheduler, and it shows, for this specific $h$, a rather flat minimum around $\beta = 2$–4. The minimum is most pronounced for the maximum delay, which is more sensitive. Notice once more the fragility of the LUc scheduler, which cannot guarantee lossless delivery with a deadline of $50T_s$ so that the maximum delay is constant at 50.

Fig. 5.7 reports the maximum and 95-th percentile of the delay of DL$_{\delta c}$/BA$_3$ELp schedulers as a function of $\delta$ for different heterogeneity factors $h$. Based on the analysis of $\beta$ in Fig. 5.6 we fixed $\beta = 3$ as a reasonable and robust value. In Chapter 4 it was
proved that DL₆c requires δ > 1 in homogeneous networks with full connectivity. The behavior of the 95-th percentiles, and indeed also of the average and other percentiles smaller than 95 not shown here for the lack of space, is minimally sensitive to the increase of δ independently of h. Another important observation is that using large postponing values δ for networks with small heterogeneity may result in large values for the maximum delay, while using 2 < δ < 10 always results in good performance.

Fig. 5.8 reports DL₆c/BAₓELp sensitivity to β of the 95-th percentile for different h. As expected, the larger h, the larger β should be to ensure good performance (remember that β represents the weight given to bandwidth information - see Equation 5.1), while a homogeneous scenario is completely insensitive to β since all peer resources are identical.
5.6 Summary

This Chapter compared a large number of scheduling strategies for P2P streaming systems in presence of network heterogeneity (peers having different upload bandwidths). Since none of the existing peer schedulers seems to perform well in all the conditions, a new peer scheduling algorithm, named BA\(\beta\)ELp has been developed and compared with the others. BA\(\beta\)ELp combined with the DL\(\delta\)c chunk scheduler is robust to heterogeneity and outperforms all the other scheduling algorithms in a large number of different conditions and scenarios.

The proposed algorithm is characterised by two tunables DL\(\delta\)c and BA\(\beta\)ELp whose optimal values depend on the network conditions. Hence, measurement based network-awareness and self-adaptation can be added to achieve dynamically optimal performance.
Chapter 5. Bandwidth-Aware Scheduling Algorithms
Chapter 6

Media-Aware Scheduling

6.1 Introduction

Until this point, we have dealt with an abstract notion of chunk (simply treating it as the unit block of data) and measured performance only in terms of chunk diffusion delay distribution, or - in another view - as percentage of chunks received before a given deadline. However, the quality perceived by the user also depends on the way chunks are generated from the media stream, on the encoding bitrate, on the audio and video codecs used, etc.

In this chapter, a more direct measure of the received media quality is used to analyze the relationship between media encoding, chunkisation policies (i.e. the way of splitting the media stream into chunks), playout delay, chunk loss, and the final received media quality.

6.2 Video Encoding and Chunkisation

Traditional P2P distribution systems generally broadcast a file to multiple peers by splitting it in fixed-size chunks. Many P2P streaming systems adopt this approach to media streaming, by dividing the encoded media into chunks. As a result, chunks generally have a fixed size in bytes, independently from the properties and characteristics of the encoded media; we refer to this way of splitting a stream in chunks as "media-unaware" chunkisation. In contrast to this approach, a media aware distribution strategy uses some knowledge about the encoded media stream to optimise its distribution and improve the streaming performance. We call this "media-aware chunkisation".

Without focusing on any particular encoding algorithm, it is possible to say that a coded audio/video stream is composed of frames, each of which starts with one or more headers and then contains the compressed data. If such headers are lost, it is not possible to decode the rest of the frame.

Another important observation is that most of the modern video encoding algorithms use three different kinds of frames: I frames (which do not depend on other frames),
Chapter 6. Media-Aware Scheduling

P frames (Predicted frames, which are coded as deltas with respect to a prediction — including motion compensation— from a previous frame), and B frames (Bi-predicted frames, which are coded as deltas with respect to a previous and a following frames). If the frame on which a P or a B frame depends is not correctly decoded, then also this frame cannot be decoded correctly, even if it has been correctly received. For this reason, losing parts of an I frame is more problematic than losing parts of a P or B frame, and if an I frame is lost, all the P and B frames directly or indirectly depending on it are damaged too. This suggests that a predicted frame should only be transmitted if the chunk containing the frames on which it depends has successfully arrived.

Summing up, the previous discussion suggests that the tolerance to chunk loss can be increased by inserting an integer number of frames in a chunk (and not splitting frames between different chunks) and by grouping predicted frames in the same chunk as their reference frames. Such a grouping can be easily performed by exploiting the regular structure of many modern video encoding algorithms which periodically repeat the same sequence of frame types (called GOP - Group Of Pictures) with a period named GOP size. Chunks containing an integer number of frames which is a multiple of the GOP size have the properties required in the discussion above. Moreover, this kind of chunkisation can reduce the variability in the chunk sizes, helping some P2P distribution algorithms that have problems in coping with variable-size chunks and making the media-aware chunkisation more efficient.

Finally, note that the term “quality” has been used very informally until now: it has been argued that a media-aware chunkisation can improve the quality when some chunks are lost, but no means have been provided to measure or quantify such quality improvements. Obviously, to verify that a chunkisation strategy can actually perform better than another one, the quality improvements need to be measured. Previous Chapters used metrics such as chunk loss ratios and delays, directly related to the chunk diffusion process. However, it is not clear how such values map to the user experience.

The next section introduces some metrics to be used as a quantitative measure of the distortion introduced in the media (the PSNR), or of the quality perceived by the end user (SSIM).\footnote{Note that audio is generally not considered because it often has a much smaller bitrate than the video.}

\[\text{GOP size} \]
6.3 Quantitative Quality Evaluation

In this Chapter, we introduce objective and quantitative metrics to measure the quality of experience P2P-TV users can have. Similar measurements have already been performed in the past for evaluating the amount of artefacts and the distortion introduced by network streaming (when some packets are lost) in encoded digital media. For example, the EvalVid [35] tool is able to evaluate the video quality loss introduced by packet loss on RTP streams, by using the PSNR or/and the SSIM as a quality metric. We use an approach similar to the EvalVid’s one, extending it to P2P streaming systems (instead of RTP).

Focusing on video, the colour of each pixel can be seen as a point in a three-dimensional space (the so called *colourspace*). Hence, it can be described by three components which depend on the used coordinate system (for example, they can be the Red, Green, and Blue components if the RGB colourspace is used, or the luminance, red chrominance, and blue chrominance components if the YCrCb or YUV colourspace is used). Each component can be coded on $Nb$ bits, and its values can be signed or unsigned. Since the PSNR is generally computed on single components, there can be many flavours of it. Here, the PSNR computed on component $c$ will be indicated as $c$-PSNR (for example, Y-PSNR is the PSNR for the luminance component in the YUV colourspace). In general, given an “original” and an “encoded/reconstructed” video frame with $m \times n$ resolution, for any component $c$ the $c$-PSNR is computed as

$$c - \text{PSNR} = 10 \log_{10} \left( \frac{\max_I^2}{\text{MSE}} \right)$$

(6.1)

where

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \| I(i, j) - K(i, j) \|^2$$

(6.2)

and $I(i, j)$ is the value of the $c$ component for pixel $(i, j)$ in the original video, $K(i, j)$ is the value of the $c$ component for pixel $(i, j)$ in the encoded/reconstructed video; $\max_I = 2^{Nb} - 1$ is the absolute maximum value for component $c$.

Y-PSNR (i.e., the PSNR computed on the luminance component) will be used as a measure of the distortion introduced by the P2P streaming system. This choice has been made for the sake of simplicity, because the luminance component is perceptively dominant, and because some preliminary tests showed that the PSNR computed on the other components (Cr and Cb) has a behaviour similar to Y-PSNR (i.e., it is affected by the chunk loss in the same way).

The Structural Similarity Index SSIM [54] is more complex than PSNR and we refer...
to the original paper for details. The idea is to represent better the errors that are perceptible and annoying to users. The SSIM index is not based on the direct difference of the components' amplitude, but it is a weighted composition of the luminance difference, the contrast difference and the difference in the structure (i.e., general organization) of the image. SSIM varies between $-1$ and $1$, but acceptable degradation normally implies $\text{SSIM} > 0.8$.

Note that when evaluating the distortion introduced by chunk loss in the received media, two problems have to be addressed:

1. A chunk generally contains multiple encoded frames. Hence, when a chunk is lost, the number of received audio or video frames is different from the original number of frames (this is the “missing frames” problem). Hence, comparing the received frames with the source can be problematic;

2. Some metrics used for evaluating the quality (for example, the PSNR) cannot cope with reconstructed frames which are identical to the original ones (in particular, if the reconstructed frame is identical to the original one, the MSE is 0, and the PSNR is undefined).

For video streams, problem 1 (missing frames) can be addressed by “filling” the
holes in the stream by a copy of the latest correctly decoded frame (in other words, if the \( n^{th} \) frame is missing, it is substituted by the last frame that has been correctly decoded). This is consistent with what many set-top-boxes do: when the stream cannot be correctly decoded, the video is not updated and a still frame is displayed. If more than one frame in a row are missing (as for instance when a large chunk is lost) the same frame is repeated over and over until the next correct frame is decoded.

Problem 2 can be addressed by evaluating the PSNR between the received media and the original uncompressed (raw) media (not the encoded media before the transmission). In this way, the MSE will never be 0, because of the effects of the encoding. This approach has the advantage that it accounts for both the effects of lost chunks (which increase when the bitrate increases) and the artefacts introduced by the encoding (which increase when the bitrate decreases). In fact, the number of lost chunks depends on the media bitrate (if the media bitrate is too high, the P2P streaming system can be overloaded and too many chunks are lost); on the other hand, reducing the bitrate to avoid lost chunks can have a bad effect too, because it introduces encoding artefacts that reduce the PSNR. Hence, some kind of trade-off is needed. The proposed way to compute the PSNR allows to find such a trade-off. Finally, this approach is consistent with the one used by EvalVid. Given a video stream \( \mathcal{V} \), both the PSNR and SSIM of the stream are computed as the average of the relative quality metric over all the frames.

Summing up, the evaluation of the quality provided by a P2P streaming system is performed as shown in Figure 6.1: first, the raw (non compressed) media stream is encoded at a specified bitrate, and the average size of a chunk is computed (note that media-aware chunkisation uses 1 GOP = 1 chunk, hence there is not much variability in the chunk sizes). The chunk size value is then used to simulate the chunks’ diffusion by using a P2P streaming simulator such as P2PTVSim [3] or SSSim [C12]. As a result, a list of lost chunks is obtained for each peer (this depends on some parameters such as the used schedulers, the playout delay, etc.). Such list is used to remove the lost chunks from the encoded stream (this is where the chunkisation algorithm is used) obtaining a corrupted stream. The missing frames are replaced with duplicates of the previous frame, and the PSNR and SSIM are finally computed by comparing the “refilled stream” with the original raw (uncompressed) stream.

### 6.4 Experimental Results

The quality evaluation techniques and tools proposed in Section 6.3 can be used to check the effects of P2P diffusion on the video quality perceived by the end user.
To focus the evaluation on the effects of chunk loss on the video quality, and to compare different chunkisation strategies, the following reference scenario has been used:

- The simple LUC/RUp (Latest Useful Chunk; Random Useful Peer — see Chapter 4 for the scheduler’s definition) scheduler has been used, simulating an overlay of 1000 peers, connected according to a random n-regular graph of degree 20;

- Each peer as well as the source has an upload bandwidth limit of 1 Mbit/s, the download bandwidth is much larger as in ADSL, and it is not a bottleneck;

- The simulations are based on the dissemination of 2000 chunks, but only the middle 100 are used for video quality evaluation;

- For the evaluation of the received video quality, the 25fps CIF “foreman” sequence has been used. The sequence has been looped 4 times, to obtain 1200 frames and 100 GOPs;

- The video is encoded with ffmpeg\(^2\) at the given target bitrate (in the first experiments, the H.264 video encoding algorithm has been used, through the x264\(^3\) codec). Details about the scripts invoking ffmpeg, and the video sequences used for these experiments are available at http://imedia.disi.unitn.it/QoE;

- Each simulation has been repeated 50 times (on different random topologies) and an average of the results has been computed.

Before looking at the PSNR results, it is interesting to understand how the chunk loss depends on the bitrate. For this purpose, Figure 6.2 shows the chunk loss rate as a function of the target video bitrate, with various target playout delays. A chunk is “lost” for a peer when it is not received within its playout time. The playout delay is expressed in “chunk times” \((T_s)\), in our case: 

\[
T_s = \frac{12\text{Frames/GOP}}{25\text{Frames/s}} = 0.48\text{s}
\]

As the video coding rate increases, the chunk size increases as well, and therefore the diffusion of a given chunk takes longer. Peers’ uplinks also get more saturated. As a consequence, the chunk loss ratio increases with coding rate at a given target playout delay.

As previously mentioned, many P2P streaming systems split the encoded stream into fixed size chunks without considering the underlying structure of the media stream. In Section 6.2 it was argued that a smarter, media-aware chunkisation can provide better

\(^2\)http://www.ffmpeg.org
\(^3\)http://www.videolan.org/developers/x264.html
6.4. Experimental Results

Figure 6.2: Chunk loss rate as a function of video bitrate and playout delay.

streaming performance in case of chunk loss. Now that a quality evaluation strategy has been presented and a reference scenario for the simulations has been set up, it is possible to confirm such intuition through a set of simulations and measurements of the video quality. In particular, a media-aware chunkisation strategy, inserting a GOP in each chunk, is compared with fixed-size chunkisation (using the average GOP size as a chunk size to achieve a fair comparison).

Figures 6.3 and 6.4 report the PSNR and SSIM at a given target playout delay, as a function of the media bitrate and the applied chunkisation scheme; all the figures show the quality (PSNR or SSIM) with no chunk loss (the quality of the encoded stream at the source, which is a function of the encoder and serves only for reference), the quality achieved with GOP sized chunks (media-aware chunkisation), and the quality achieved when using fixed size chunks (media-unaware chunkisation). This allows understanding the impact of loss ratio on the video quality. For example, the top plot of Fig. 6.3 shows what happens if the playout delay is set relatively low ($12T_s = 5.76$ s): in this case, as soon as the chunk loss ratio starts to increase (at bitrate 0.5 Mbit/s, according to Fig. 6.2) the quality of the received stream becomes much smaller than the quality of the original encoded stream. In case of media-aware chunkisation, the PSNR is always much higher than for media-unaware chunkisation.

Increasing the playout delay to $14T_s$ creates an interesting situation, showing that for media-aware chunkisation the PSNR can be increased by increasing the bitrate up to about 0.8 Mbit/s (in case of media-unaware chunkisation, increasing the bitrate over
0.7 Mbit/s does not increase the PSNR, and increasing it over 0.8 Mbit/s even decreases the PSNR. Finally, the bottom parts of the figures show the quality achieved with a large playout delay (26T_s), which reduces the chunk loss (see Fig. 6.2) to a minimum and allows to achieve high video quality by increasing the bitrate to around 0.95Mbit/s.

In any case, by comparing all the figures it is possible to see that media aware...
chunkisation provides significantly higher quality (both in PSNR and SSIM) than fixed size chunks. Moreover, while in case of no chunk loss the PSNR increases with the bitrate, the distribution system cannot keep up with the increased amount of traffic maintaining the playback delay constraint.

Figure 6.5 analyses the effects of chunk and peer scheduling on the video quality,
Figure 6.5: Chunk loss rate and PSNR with different schedulers and GOP sized chunks, as a function of playout delay.

showing the chunk loss rate and PSNR values obtained with different schedulers (namely LUc/RUp, LUc/ELp, and DLc/ELp). The analysis of the two plots immediately shows the impact of the loss rate on the PSNR, but also shows that the loss rate is a more sensitive measure: the PSNR rises within 2 dB of the encoded sequence as soon as the loss rate falls below 10%. Quality at higher loss rates degrades sharply. Thus a fast feedback from the network level identifying potentially dangerous situations where the loss rate starts increasing say, above 1% can prevent quality degradation that will be difficult to recover. The second observation is the impact of the scheduler: a good scheduler (in this case DLc/ELp) ensures lossless delivery with low delay (in this example 13 $T_s$ which is less than 7 s), and guarantees unaffected PSNR (and quality) to the users. As a result of this analysis, it is possible to state that combining a media-aware chunkisation (in
6.5. Summary

This Chapter described a methodology for evaluating the video quality in P2P streaming systems, complementing “network-oriented” performance metrics such as the chunk diffusion delay or the chunk loss rate. The proposed approach is based on using a P2P system simulator for identifying the chunks lost during the stream diffusion. The simulator is then interfaced to a tool that can actually remove the lost chunks from an encoded media stream to obtain a “received stream” which can be compared with the original one. The comparison is based on quantitative quality metrics such as PSNR and SSIM.

The proposed methodology allows to evaluate the effects of media encoding, chunkisation, scheduling, and network condition on the quality perceived by the end user.
As an example, the quality improvement due to media-aware chunkisation has been evaluated.

In Chapter 7, more advanced media-aware chunkisation strategies will be studied using this methodology.
Chapter 7

Deadline-Based Sub-Stream Scheduling

Splitting a P2P video distribution in multiple media flows with different priorities is an interesting approach for developing flexible and adaptive P2P streaming systems. Such an approach can both yield satisfactory quality to all end users and be light in network resources usage, because low-priority flows can be discarded when target peers do not have enough resources to receive them.

An extension of the DLc scheduler is proposed, where different flows of chunks are prioritized using different deadline postponing parameters for each flow. Some experimental results show good differentiation properties and much better streaming performance than without priorities or with strict priority enforcement. Also, PSNR measures on real video streams show improvements compared to both strict priority and single stream distribution.

7.1 Extending DLc to Sub-Stream Scheduling

The key idea of DLc scheduling is the association of a scheduling deadline $d_c$ to every chunk instance. Such a deadline is postponed by a constant value $\delta$ every time the chunk is transmitted to a neighbour. When a chunk has to be sent, the one having the earliest scheduling deadline is selected, according to the EDF (Earliest Deadline First) algorithm.

The DLc scheduler can be extended to sub-stream scheduling by assigning each sub-stream a different deadline postponing parameter. More formally, let $\mathcal{H} = \bigcup_{s=1}^{N_s} \mathcal{H}_s$ be the set of all chunks composing the stream where $N_s$ is the total number of sub-streams, $\mathcal{H}_s$ is the set of chunks of sub-stream $s$; $c_{s,h}$ denotes the $h$-th chunk of sub-stream $s$, and $d_{s,h}$ its deadline. Let also $\delta_s$ be the deadline postponing parameter for chunks in sub-stream $s$, and $\mathcal{C}'(P_i, t)$ be the set of chunks at peer $P_i$ that are needed by some of the peers within its neighbourhood $\mathcal{N}_i$ (i.e., by at least one of the peers that $P_i$ is exchanging buffermaps with), at the time $t$ of the scheduling decision. The chunk
scheduling strategy at peer $P_i$ is the following:

\[
\begin{align*}
\text{Update } & C'(P_i, t) \\
& c_{s,k} : \forall c_{r,h} \in C'(P_i, t), d_{s,k} \leq d_{r,h} \\
& d_{s,k} := d_{s,k} + \delta_s \\
& \text{Transmit } c_{s,k}
\end{align*}
\] (7.1)

The deadlines are initialised by the source to the chunk generation time (the time when the source emits the chunk), and are postponed by $\delta_s$ every time a chunk instance is sent. Note that the priority of a chunk instance depends on the postponing parameter $\delta_s$ of its sub-stream $s$ and on the number of times it has been transmitted. Hence, the postponing parameter $\delta_s$ permits to control the importance of a sub-stream: $\delta_s < \delta_r$ implies that sub-stream $s$ has a higher priority than sub-stream $r$. Intuitively, if $\delta_r = \delta_s$, then the two sub-streams are considered equally important, while if $\delta_r - \delta_s \gg T$, almost strict priority is guaranteed to chunks belonging to sub-stream $s$ in all peers except for the source. Hence, the sub-streams models previously presented in literature (all sub-streams are equally important, or strict priority order) are special cases of this more generic model.

One of the key features of the distributed DLc scheduling policy is that the priority of each chunk at each peer is a function of the chunk instance history, i.e., the more the chunk has been duplicated and diffused into the system, the lower its priority. This feature makes DLc different from scheduling algorithms that simply prioritise the different sub-streams. In fact, according to this scheduling algorithm chunks with small $\delta_s$ will be transmitted faster, thus decreasing their priority because of many duplications, so that also some part of the lower priority sub-streams (large $\delta_s$) will be diffused (avoiding starvation).

As shown in previous chapters, the DLc chunk selection strategy can be coupled with any suitable peer selection strategy either in chunk-first or in peer-first mode. Clearly, performance will depend on both the chunk and the peer selection algorithm. In what follows, we use chunk-first selection with ELp peer selection, but most of our considerations apply to the use of DLc with other peer schedulers as well.

As already mentioned, DLc is based on the EDF scheduling algorithm, derived from the seminal work of Liu and Layland [36]; however, dynamic and distributed deadlines change the system properties non-marginally. In a multi-stream system the emission of chunks by the source is not necessarily periodic, and different sub-streams can have different priority and different emission periods. In this context a proof of optimality like the one in Chapter 4 is not feasible.
Finally, each chunk instance embeds both its scheduling deadline and the deadline postponing parameter, hence peers do not need to be aware of the sub-streaming details (not even about the total number of sub-streams composing the stream). This gives the freedom to dynamically change the number of sub-streams and also their importances (in a simple priority-based scheme, this would be more difficult).

## 7.2 Evaluation Scenarios

The performance of P2P streaming systems has been evaluated using the hybrid approach of Chapter 6: the sub-stream handling mechanism (splitting a video stream in multiple sub-streams, and dividing such sub-streams in chunks, as well as the inverse process to rebuild the stream) has been implemented, but the chunk loss process is obtained via simulation. Then, lost chunks are removed from the encoded stream, and a “corrupted stream” (as it would have been received by a remote peer) is constructed, decompressed, and compared with the original one.

Chunk loss patterns have been derived by simulating the diffusion of $M_c = 2000$ chunks over $N = 1000$ peers connected by random topologies with neighbourhood size $N_N = 20$. Simulations are run over 100 different topologies generated at random, and the results are mediated over all runs. As in previous chapters and in [17], chunks are diffused by considering a system where peers exchange buffer maps with neighbours to generate a distributed common knowledge of the system, and the chunks are actively pushed from peers owning them to peers that need them.

The source generates $N_s$ sub-streams labelled in order of importance: $s = 1 \succ s = 2 \ldots \succ s = N_s$, with $\succ$ indicating that the left hand side is ‘more important’ than the right hand side.

Before evaluating the real performance of the proposed algorithm, Sect. 7.3 presents some simple simulations to verify the correctness of sub-stream handling and to show the basic properties of the scheduler. After that, Sect. 7.4 evaluates the performance of a realistic sub-stream composition, using a real video stream and a more complex network setup with heterogeneous peers having different access bandwidths.

## 7.3 Algorithm Validation

To verify that the proposed scheduling algorithm can correctly differentiate the sub-streams, consider a homogeneous network where all the peers have unlimited download capacity, and an upload bandwidth equal to the stream bitrate. All chunks are identical
in size, and chunks belong to $N_s$ sub-streams according to the following rule:

$$\mathcal{H}_s = \{c_i : i \mod N_s = s\}$$

### 7.3.1 Sub-Stream Differentiation

First the sensitivity to the deadline postponing $\delta_s$ parameter is analysed with $N_s = 2$. Results are compared with other means of differentiating sub-streams, namely we compare:

- **DL$^*$c** priority using DLc fixing $\delta_1 = 5$ and modifying $\delta_2$;
- **PLUc** giving strict priority to $s = 1$ and using a Latest Useful chunk selection among chunks of the same sub-stream;

Chunk losses in this experiment are due to chunks delivered later than the playout time. Fig. 7.1 shows the chunk loss rate as a function of the various schedulers and different values of $\delta_\Delta = \delta_2 - \delta_1$. Note that when $\delta_\Delta = 0$, DL$^*$c does not differentiate the two streams and their chunk loss rates are identical. This corresponds to the loss rate of DLc without prioritization, and it is shown in the leftmost side of the figure (the small piece with linear scale on the x axis). PLUc is not parametrised, therefore its performance is independent of $\delta_\Delta$.

The lowest average chunk loss ratio is provided by DLc, and it increases gradually as prioritization is increased in DL$^*$c (the right part of the figure is in semi-log (x) scale). At the same time, the spread between sub-stream 1 and 2 opens, showing that
Figure 7.2: loss ratio and maximum delay of chunks belonging to different sub-streams, using 4 sub-stream encoding

prioritization in the overlay works. The slight increase in the average is the price paid to support differentiation.

From the figure, it is possible to see two advantages of DL\textsuperscript{c} over PLU\textsubscript{c}: i) it provides a considerably lower loss rate (see for instance $\delta_2 - \delta_1 = 2$); ii) it is more tunable and configurable, allowing to select interesting trade-offs between the importance of the two streams (when $\delta_2 - \delta_1 < 1$).

7.3.2 Robustness to Chunk Loss

Priority should also protect from random losses and errors. In this experiment a simple Bernoulli process marks chunks as errored with probability $p$, reported on the x axis in Fig. 7.2. Other simulation parameters are the same as in the previous tests, except for $N_s = 4$, and the source’s bandwidth which is increased to 5 to enable recovery of chunks lost in the first transmission. The network is overloaded due to the chunk re-transmissions so we cannot expect all chunks to be delivered to all peers. The upper plot reports the average chunk loss rate at peers (chunks arriving after playout time), while the lower plot reports the worst case diffusion delay, limited to 32 by the playout deadline. For simplicity, $\delta_1 = 5$ and $\delta_\Delta = \delta_s - \delta_{s-1} = 5$. The figure shows that higher
Chapter 7. Deadline-Based Sub-Stream Scheduling

![Graph showing chunk size variation over the first 100 chunks]

Figure 7.3: Chunk size variation over the first 100 chunks: 31.7% of the bytes are in I chunks, 46.9% are in P chunks and 21.4% are in B chunks.

Priority sub-streams are favoured in recovery from errors and are transferred with low losses and delays, even if errors equally affect all the sub-streams. In fact, the last figure shows that the most important sub-stream can be delivered reliably and with low delay even if the network discards nearly 50% of the chunks.

7.4 Performance Evaluation

The performance of the proposed strategy is evaluated by considering a sub-stream model for encoded video streams composed of different types of frames (I, P, and B as in the MPEG standards). The stream is split in 3 sub-streams by generating 3 chunks for each Group of Pictures (GoP): a chunk containing the I frame, a chunk containing all the P frames, and a chunk containing all the B frames of the GoP. The video has been encoded at 1Mbit/s using the MPEG4 codec provided by ffmpeg, and a GoP size of 12 frames. Figure 7.3 shows the evolution of the chunk size for the three sub-streams with time. Note that chunks for sub-stream 1 contain 1 I frame, while chunks from sub-stream 2 (and 3) contain multiple P (and B) frames; as a result, chunks from sub-stream 2 can be larger than chunks from sub-stream 1.

Figure 7.4 plots the chunk loss rate for the three sub-streams, as a function of prioritisation, and shows how tuning $\delta$ values allows the dynamic adaptation of losses in different sub-streams (even if, for the sake of brevity, we only report the case when the $\delta$ increment between classes is the same - the x axis shows $\delta_{\Delta}$, normalised w.r.t. $T=0.16$ s). The right part of the plots shows that $DL^*c$ with high $\delta$ values provide a performance similar to PLUc, and $DL^*c$ slightly outperforms PLUc.

---

1 We use the standard “foreman” sequence concatenated several times, encoded to IBPBPBPBPBPB GoP structure with ffmpeg using “-f m4v -vcodec mpeg4 -g 12 -bf 1 -b 1M”.
The properties of this sub-streams model can be better highlighted by considering a more realistic network model, in which peers have different access bandwidths. In more details, peers belong to three classes with different access bandwidths, and each class has symmetric links with equal upload and download limits: high-bandwidth peers have 2 Mbit/s, mid-bandwidth peers have 1 Mbit/s and low-bandwidth ones have 0.5 Mbit/s. The fraction of high- mid- and low-bandwidth peers are $1/6$, $3/6$, and $2/6$ respectively; as a result, the average bandwidth is 1 Mbit/s.

Figure 7.5 shows the chunk loss rate per peer class (note the different scale on the y axes). Losses at high-bandwidth peers are marginal, while low-bandwidth peers can loose almost 100% of sub-stream 3. For high-bandwidth peers, DL*c outperforms PLUc all over the scale. Mid-bandwidth peers loose some chunks of sub-stream 1 for small $\delta_{\Delta}$ values. For $\delta_{\Delta} \geq 5$, DL*c performs equal or better than PLUc in all sub-streams. Unsurprisingly, performance of these peers resembles that of Fig. 7.4. Finally, for low-bandwidth peers, the three sub-streams suffer different losses even without prioritisation (leftmost point of DL*c curves), due to differences in chunk size. Sub-stream reception can be highly tuned by selecting the value of $\delta_{\Delta}$.

### 7.4.1 Received Video Quality

To understand the impact of lost chunks when sub-streaming is used, the quality of the video received by individual peers has been evaluated by comparing the received stream

---

2 Larger chunks have higher transmission times, but can also postpone the transmission of following (smaller) chunks, hence the effect of chunk size is controversial. There is no clear correspondence between chunk size and chunk loss.
Chapter 7. Deadline-Based Sub-Stream Scheduling

Figure 7.5: Chunk loss for various peer classes and chunk priority classes

Figure 7.6: per frame PSNR values with sub-streams (three separate chunks for the I,P and B frames of a GoP) and without sub-streams (1 chunk = 1 GoP)
7.4. Performance Evaluation

![Figure 7.7: PSNR for all peers, and at different peer classes as a function of prioritization; average for all peers; high-, mid- and low-bandwidth peers](image)

![Figure 7.8: PSNR increase changing from PLUc to DL\textsuperscript{*}c in different peer classes](image)
with the original YUV encoded sequence. PSNR (Peak Signal-to-Noise Ratio) values are calculated and averaged over all decoded frames using the methodology introduced in Chapter 6.

Figure 7.6 provides some insight into how chunk losses affect PSNR values of individual frames. When sub-streams are not used and chunks are formed e.g. per GoP, the loss of a chunk pauses the video for 12 frames (about 0.5 seconds), degrading the video quality (PSNR) significantly. With sub-streams, when chunks containing B frames are lost, the effective frame rate is reduced to half, making the playback more rough. Note that PSNR values drop significantly even for the loss of one frame, but this change is less noticeable to human eyes (especially if audio playback remains continuous). In what follows, for simplicity, we use the average of PSNR as our video quality metric.

Figure 7.7 shows the average PSNR at peers of different bandwidth classes. Since the system is not overloaded (1 Mbit/s encoding), peers that have enough bandwidth receive the whole stream with small losses (mainly due to chunk size variation and delay constraints).

For high-bandwidth and mid-bandwidth peers, DL∗c outperforms PLUc by 0.5–1 dB in all cases. For low-bandwidth peers (which end up experiencing most of the chunk losses), using a small δΔ (which means no sub-streams differentiation) results in bad video quality (hence, sub-stream differentiation is important — note that prioritisation allows to gain almost 4dB). Increasing δΔ, the PSNR for class 3 increases, and for δΔ = 10 it is just a little bit worse than the PSNR achieved using PLUc. However, the quality of the video received by the other peers is much higher with DL∗c than with PLUc. Looking at the average between different peer classes (top of Figure 7.7), one can notice that DL∗c with δΔ > 2 always outperforms PLUc. Figure 7.8 shows the PSNR improvements achieved by using DL∗c, proving that the proposed sub-stream scheduling strategy can outperform strict priorities.

Finally, some experiments have been repeated changing the video codec and the video bitrate. The results obtained by using H.264 (in particular, the x264 codec) are consistent with the results obtained using MPEG4, and are shown in Figure 7.9. The figure compares schemes without prioritisation (DLc), with strict prioritisation (PLUc) and DL∗c(fixing δΔ = 9 for DL∗c) either in underload (bitrate < 1Mbps) and overload.

---

3Such values highly depend on the codec, as well as on the error concealment methods used in decoding. We use ffmpeg decoding substituting entirely missing frames with the last correctly decoded frame.

4We remind the reader that the average of per-frame PSNR values indicates quality, but it does not map well to a MOS like evaluation. Visual inspection of the reconstructed videos showed significantly better quality for sub-streams (with priority) than without sub-streams.
Figure 7.9: PSNR with different schedulers, at different peer classes, as a function of video encoding rate

(bitrate > 1Mbps) conditions. When the system is underloaded, high-bandwidth peers receive the entire stream. Video quality at bandwidth limited peers is largely improved by prioritization (see difference between lowest curve and the one above). However, even with prioritization it decreases as the encoding rate goes above the bandwidth limit: the simple sub-streaming scheme used to create the chunks based on frame type cannot guarantee constant PSNR. When, instead, the system is overloaded, chunk prioritization and sub-streaming can still improve performance, and DL∗c clearly outperforms PLUc.

7.5 Summary

This Chapter introduced the idea of deadline-based sub-stream scheduling in P2P streaming systems, based on scheduling deadlines that are dynamically updated upon chunk transmission. Using this mechanism, the priority of a chunk depends not only on its sub-stream, but also on the number of times it has been replicated. Some previous sub-stream models (equally important sub-streams, and strict prioritisation) are special cases of the proposed model.

Experimental results look promising, yielding a general framework for defining chunk scheduling algorithms that are able to differentiate between flows, sub-streams, or even single chunks, in multimedia diffusion, enabling low-latency video distribution. As shown through experiments on real video sequences, the proposed algorithm can outperform both strict priorities and equally important sub-streams.
Chapter 8

Conclusions

We have tackled two largely different areas of overlay networks in the thesis, namely Anonymous Routing and live P2P Streaming.

Starting from fundamental results on the detailed performance characterization of TCP flows, we have arrived to the conjecture that the use of TCP as the tunneling technology in some overlay network designs could largely hinder performance. Based on this observation, we have designed IPpriv, a new Anonymous Routing system that builds entirely on a datagram service. Besides being datagram based, IPpriv also has other characteristic features, most important being that its data-plane operation is entirely built on standard tunneling and encryption technologies.

We have confronted IPpriv with the state of the art Tor system through controlled experiments. Care had been taken to isolate various factors contributing to performance by running experiments both in our lab and over the Internet. Our measurements have clearly demonstrated that the new design provides significant improvements in transmission delays.

In the case of live P2P Streaming, we concentrated our attention on the scheduling problem of mesh type chunk-based systems. First, the new DLc/ELp scheduling algorithm had been introduced, and we have formally proven that this algorithm achieves strict optimality under idealized conditions. According to the best of our knowledge, this is the first distributed algorithm to achieve such bound.

In following chapters, several assumptions of the idealized model had been relaxed. First, connectivity constraints were introduced, then bandwidth heterogeneity was considered. The effects of chunk formation policies was also studied. Finally, based on the structure of the media stream, a new chunk priority scheme was introduced.

Simulation studies show that DLc/ELp and its bandwidth- and media-aware variants (BAELp, DL*c) outperform other algorithms from literature in a large variety of settings.
8.1 Summary of new results

The following eight claims summarize the most important new results obtained in the thesis. References to related publications are also shown for each claim.

8.1.1 Performance of Anonymous Routing Overlays

Claim 1.1. [J1, C4] I have proposed a method for the calculation of the completion time distribution of short-lived TCP connections based on OMQN (open multiclass queuing network) models of protocol behavior. (see Section 2.4 of the thesis)

Claim 1.2. [J1, C4] I have shown — through ns-2 simulations — that the model predicts TCP connection completion times with high precision in various networking scenarios. The proposed technique is computationally efficient, and its asymptotic complexity does not depend on the network topology, on the number of concurrent flows, and on other network parameters. (see Section 2.5 of the thesis)

Claim 1.3. [C3, C5, C6] I have proposed a new system design for anonymous routing overlays, based entirely on standard IPsec functionality, thus avoiding bottlenecks found in other systems due to the use of TCP tunnels. (see Section 3.3 of the thesis)

Claim 1.4. [C3, C5] I have implemented the design of Claim 1.3 and confronted its performance with a TCP based implementation in both emulated and real environments, demonstrating that end-to-end delays can be largely improved in anonymous routing systems. (see Section 3.4 of the thesis)

8.1.2 Performance of Peer-to-Peer Streaming Overlays

Claim 2.1. [B1] I have proposed the new Earliest-Latest peer scheduling algorithm (ELp) for P2P streaming systems, and proved that — in combination with the well-known Latest Useful chunk scheduler (LUs) — it achieves strict optimality in idealized conditions. (see Section 4.4 of the thesis)

Claim 2.2. [B1] I have shown that the ELp algorithm still achieves strict optimality in combination with a whole class of Deadline-based chunk scheduling algorithms (DLc) in the same idealized conditions. This combination is also robust to impairments when relaxing some idealistic assumptions of the mathematical model. (see Section 4.5 of the thesis)

Claim 2.3. [C2, C1] I have proposed a network-aware extension of the algorithm of Claim 2.1, and analyzed its performance in heterogeneous network scenarios. The new
BAELp algorithm is equivalent to ELp in the homogeneous case, and it outperforms other known algorithms in the heterogeneous case. (see Chapter 5 of the thesis)

Claim 2.4. \[C7, C8\] Based on the algorithm of Claim 2.2, I have proposed a novel media-aware scheduling algorithm that assigns priorities to chunks based on their media content. I have shown that the new algorithm improves overall system performance. (see Chapters 6 and 7 of the thesis)

8.2 Applicability of results

Results on TCP performance are applicable in the performance analysis of a wide range of systems. In fact, TCP connections are being used in a large variety of ways, often as a consequence of using higher level programming libraries, without even knowing what is underneath. It is enough to mention the fast opening and closing of TCP connections in some P2P file sharing applications, the use of several parallel connections in browsers, or various HTTP based IPC mechanisms often used in composite web services to demonstrate that detailed characterization of TCP’s short-term behavior is essential for understanding system behavior in some of today’s complex systems.

IPpriv presents a fast and efficient datagram based anonymous system design. On the one hand, it has some applicability limitations: namely, it does require system and network administrators to enable IPsec policies in some selected routers. On the other hand it also conveys two important messages: first, many of the performance bottlenecks attributed to ”routing around the world” are not in fact due to the increased length of the transmission path, but rather to the misguided selection of underlying overlay techniques. Second, it presents a realistic alternative to end-user based anonymous routing overlays, providing techniques that can anonymize traffic in the network itself with standard carrier grade routing equipment, without introducing application level routing components.

Results on live P2P Streaming are not only applicable, but they are already being applied in the open-source PeerStreamer streaming framework. The results presented and their improvements are also being discussed in the IETF PPSP working group, and will hopefully influence the design of the future peer-to-peer streaming protocol standards.
Bibliography


Related Publications


