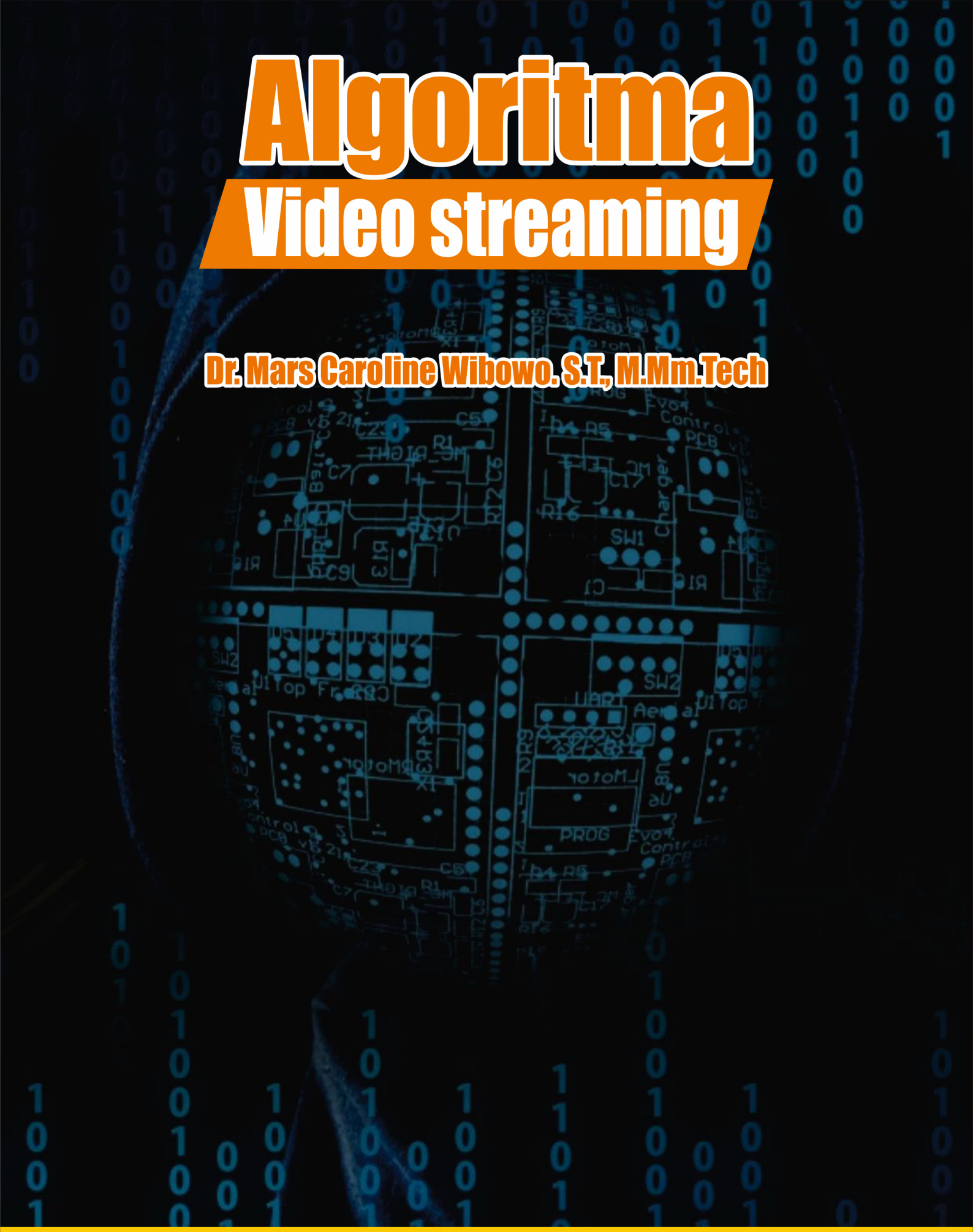
YAYASAN PRIMA AGUS TEKNIK

Algoritma Video streaming

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Algoritma Video streaming

Dr. Mars Caroline Wibowo. S.T, M.Mm.Tech



YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
AYASAN PRIMA AGUS TEKNIK

Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8120-54-3 (PDF)

Algoritma Video Streaming

Penulis :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

ISBN : 9 786238 120543

Editor :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Penyunting :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Segala puji syukur bagi Tuhan Yang Maha Esa atas rahmat dan karunia-Nya , sehingga penulis dapat menyelesaikan buku ajar berjudul ***"Algoritma Video Streaming***" dengan lancar. Adapun buku ini yang telah selesai kami buat secara semaksimal dan sebaik mungkin agar menjadi manfaat bagi pembaca yang membutuhkan informasi dan pengetahuan. Buku ini mengusulkan algoritma untuk pengiriman video adaptif yang meningkatkan QoE penampil. Algoritme mudah diterapkan dan kami menerapkannya di dash.js, pemutar referensi DASH. Penyedia video menggunakan algoritme kami dalam setelan produksi. Kami juga mengembangkan dua testbed open source untuk mensimulasikan algoritme video adaptif, satu untuk video 2-D tradisional dan satu lagi untuk video 360°.

Buku ini juga merancang dan mengimplementasikan Sabre, alat sumber terbuka yang tersedia untuk umum yang dapat digunakan oleh peneliti untuk menjalankan simulasi algoritme ABR yang akurat menggunakan arsitektur pemutar yang mirip dengan dash.js. Dalam buku ini menggunakan Sabre untuk merancang dan menguji BOLA-E dan DYNAMIC, dua algoritme yang menyempurnakan algoritme BOLA berbasis buffer ABR. Penulis juga mengembangkan algoritme FAST SWITCHING yang dapat menggantikan segmen yang telah diunduh dengan segmen dengan kecepatan bit lebih tinggi.

Penulis juga membandingkannya secara langsung. BOLA-E sedikit lebih baik daripada DYNAMIC ketika bandwidth jaringan memiliki variabilitas yang lebih tinggi, situasi yang juga sangat menantang untuk algoritma THROUGHPUT. Di sisi lain, DYNAMIC sedikit lebih baik untuk kapasitas buffer kecil, situasi yang dapat menjadi tantangan untuk semua algoritme berbasis buffer termasuk BOLA-E. Penulis merancang ABR*, algoritme ABR dalam sistem VOXEL yang menggunakan keandalan transpor selektif untuk menghindari buffering ulang tanpa dampak signifikan pada QoE. Penulis merancang dan mengimplementasikan Sabre360, tempat pengujian simulasi untuk algoritme adaptasi video 360°.Dan mendemonstrasikan bagaimana Sabre360 dapat menjadi alat yang berguna untuk merancang dan membandingkan algoritme baru.

Demikian buku ini dibuat, dengan harapan agar pembaca dapat memahami informasi dan juga mendapatkan wawasan mengenai Algoritma konten video, serta dapat bermanfaat bagi penulis dan pembaca.

Semarang, Juli 2023
Penulis,

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

DAFTAR ISI

BAB 1
PENDAHULUAN

Video online adalah aplikasi "pembunuh" Internet dengan video saat ini menyumbang lebih dari setengah lalu lintas Internet. Kepemirsaan video tumbuh dengan kecepatan tinggi dan video diharapkan mencapai lebih dari 85% dari semua lalu lintas Internet dalam beberapa tahun. Karena semua bentuk media tradisional bermigrasi ke Internet, penyedia video menghadapi tantangan yang menakutkan dalam memberikan kualitas pengalaman (QoE) yang baik bagi pengguna yang menonton video mereka. Penyedia video beragam dan mencakup perusahaan media besar (seperti: MNC dan TransMedia), outlet berita (misalnya CNN), organisasi olahraga, dan layanan berlangganan video (misalnya, Netflix, Hulu). Penelitian terbaru menunjukkan bahwa video berperforma rendah yang dimulai dengan lambat, diputar dengan bitrate lebih rendah, dan sering macet dapat menyebabkan pemirsa meninggalkan video atau menonton lebih sedikit menit video, secara signifikan mengurangi peluang untuk menghasilkan pendapatan bagi penyedia video, menggarisbawahi perlunya pengalaman pengguna yang berkualitas tinggi.

1.1 STREAMING VIDEO ADAPTIF

Memberikan pengalaman berkualitas tinggi untuk pengguna video membutuhkan penyeimbangan dua persyaratan kontras. Pengguna ingin menonton versi video dengan kualitas tertinggi, di mana kualitas video dapat diukur dengan kecepatan bit saat video dikodekan. Misalnya, menonton film dalam definisi tinggi (HD) yang dikodekan pada 10 Mbps bisa dibilang memberikan pengalaman pengguna yang lebih baik daripada menonton film yang sama dalam definisi standar (SD) yang dikodekan pada kecepatan bit 800 kbps. Faktanya, terdapat bukti empiris bahwa pengguna lebih terlibat dan menonton lebih lama saat video disajikan pada bitrate yang lebih tinggi. Namun, tidak selalu memungkinkan bagi pengguna untuk menonton video pada bitrate tertinggi yang disandikan, karena bandwidth yang tersedia pada koneksi jaringan antara pemutar video pada perangkat pengguna dan server video membatasi bitrate yang dapat ditonton. Bahkan, memilih bitrate yang lebih tinggi dari bandwidth jaringan yang tersedia akan menyebabkan video macet di tengah pemutaran, karena kecepatan pemutaran video melebihi kecepatan pengunduhan video. Pembekuan video semacam itu disebut rebuffer dan pemutaran video terus menerus tanpa rebuffer adalah faktor kunci dalam QoE yang dirasakan oleh pengguna. Oleh karena itu, menyeimbangkan persyaratan kontras untuk memutar video pada bitrate tinggi sementara pada saat yang sama menghindari buffering ulang merupakan hal penting untuk memberikan pengalaman menonton video berkualitas tinggi.

Kebutuhan untuk menyesuaikan pemutaran video dengan karakteristik perangkat dan jaringan telah menyebabkan evolusi streaming *Adaptive Bitrate* (ABR) *dan HTTP Adaptive Streaming* (HAS) yang sekarang menjadi standar de facto untuk pengiriman video di Internet. Streaming ABR mengharuskan setiap video dipartisi menjadi beberapa segmen, di mana setiap

segmen sesuai dengan beberapa detik pemutaran. Setiap segmen kemudian dikodekan dalam sejumlah bitrate yang berbeda untuk mengakomodasi berbagai jenis perangkat dan konektivitas jaringan.

Ada beberapa implementasi populer dari sistem streaming HAS, termasuk *HTTP Live Streaming* (HLS) Apple, Live Smooth Streaming (Smooth) Microsoft dan Adaptive Streaming (HDS) Adobe. Masing-masing memiliki penerapannya sendiri dan sedikit modifikasi pada teknik dasar ABR yang dijelaskan di atas. Perkembangan utama baru-baru ini adalah penyatuan standar sumber terbuka untuk streaming ABR yang disebut MPEG-DASH. DASH secara umum mirip dengan protokol ABR lainnya dan merupakan fokus khusus dalam evaluasi empiris dalam buku ini. Forum Industri DASH juga menyediakan pemutar video referensi sumber terbuka, dash.js.

Baru-baru ini, realitas virtual dan augmented reality telah mendapatkan popularitas. Konten video 360° dapat ditonton oleh penonton menggunakan head-mounted display (HMD) dan merespons gerakan penonton. Memberikan pengalaman berkualitas tinggi untuk video 360° lebih kompleks daripada video 2-D. Video mencakup area tampilan yang lebih besar dan dengan demikian membutuhkan bitrate yang lebih tinggi. Untuk menambah persepsi kedalaman, video perlu ditransmisikan dalam stereo, yang selanjutnya meningkatkan bitrate keseluruhan. Penyedia video dapat mengurangi kebutuhan bandwidth dengan mengeksploitasi fakta bahwa area pandang, yaitu area video yang terlihat oleh pengguna, hanyalah sebagian dari area 360° yang dikodekan dalam video. Penyedia dapat mempartisi area video dalam petak dan menyandikan petak secara terpisah. Pemutar video 360° hanya perlu mengunduh petak yang ada di viewport. Pemutar video memerlukan algoritme prediksi tampilan untuk memprediksi ubin mana yang akan diunduh. DASH memiliki ekstensi deskripsi hubungan spasial (SRD) yang mendukung petak untuk video 360°.

Metrik ABR untuk QoE tinggi

Buku ini memulai penelitian ABR dalam buku ini dengan mendapatkan umpan balik ekstensif dari penyedia video dan pengguna di seluruh spektrum industri media. Secara khusus, memanfaatkan panggilan konferensi dua mingguan oleh komunitas dash.js untuk umpan balik. Hampir ada konsensus tentang persyaratan untuk algoritme ABR yang baik yang dinyatakan di bawah ini.

1. Bitrate Tinggi. Harus memutar video dengan kualitas berkelanjutan tertinggi (yaitu, kecepatan bit).
2. Penyangga Ulang Rendah. Harus menghindari peristiwa rebuffering (yaitu pembekuan) yang terjadi karena buffer klien kosong.
3. Osilasi Rendah. Harus menghindari osilasi kecepatan bit yang berlebihan di mana kualitas video sering diubah selama pemutaran.
4. Responsif terhadap Acara Jaringan. Harus bereaksi cepat terhadap peristiwa jaringan. Misalnya, jika throughput jaringan tiba-tiba turun, algoritma ABR harus menurunkan bitrate video untuk menyesuaikan dengan status jaringan baru.
5. Responsif terhadap Acara Pengguna. Harus bereaksi cepat terhadap peristiwa pengguna. Misalnya, jika pengguna memulai video baru, atau mencari tempat baru dalam video yang

sama, pemutaran akan mulai diputar dengan cepat pada kecepatan bit berkelanjutan tertinggi.

6. Streaming Langsung Latensi Rendah. Harus bekerja dengan baik saat streaming video langsung yang membutuhkan latensi rendah, di mana latensi adalah waktu maksimum antara saat video diambil dan saat pengguna melihatnya. Tantangan utamanya adalah karena latensi harus rendah, buffer klien harus kecil dan tidak dapat menampung lebih dari beberapa segmen. Dengan demikian, segmen video tidak dapat diambil oleh klien jauh sebelum diputar. Buffer kecil menyisakan sedikit ruang untuk kesalahan karena satu keputusan ABR suboptimal dapat mengakibatkan terkurasnya *buffer*, yang mengakibatkan buffering ulang. Definisi tepat latensi rendah bersifat subyektif dan bergantung pada kasus penggunaan. Dalam buku ini, latensi rendah yang dimaksud adalah latensi di bawah 10 detik.

1.2 TANTANGAN STREAMING VIDEO ADAPTIF

Mencapai QoE yang tinggi untuk streaming video merupakan tantangan besar karena banyaknya perangkat berkemampuan video yang mencakup smartphone, tablet, desktop, dan televisi (Gambar 1.1). Selanjutnya, perangkat itu sendiri dapat dihubungkan ke Internet dengan berbagai cara, termasuk kabel, serat, DSL, WiFi, dan nirkabel seluler, masing-masing memberikan karakteristik bandwidth yang berbeda. Dalam buku ini mencantumkan empat tantangan streaming adaptif yang ditangani dalam disertasi ini dan menguraikan solusi yang diusulkan.

Gambar 1.1. Streaming video di berbagai perangkat melalui beragam kondisi jaringan.

Pendekatan berprinsip untuk streaming kecepatan bit adaptif

Tantangan adalah berbagai macam algoritme ABR yang umumnya termasuk dalam salah satu dari dua kategori, berbasis throughput dan berbasis buffer. Algoritma berbasis throughput mengestimasi throughput jaringan dan memilih bitrate video yang tidak melebihi estimasi throughput. Algoritma berbasis buffer memilih bitrate berdasarkan seberapa banyak video yang diunduh sudah tersedia di buffer video. Namun, algoritma di kedua kelas secara tradisional heuristik tanpa pembenaran yang berprinsip.

Dalam buku ini merumuskan streaming video sebagai masalah pengoptimalan menggunakan pengoptimalan Lyapunov dengan dua tujuan utama, kecepatan bit tinggi dan buffer ulang rendah. Dalam penelitian ini memperoleh (*Buffer Occupancy based Lyapunov Algorithm*) BOLA, algoritma ABR berbasis buffer. Dalam buku ini tidak secara khusus merancang BOLA untuk berbasis buffer, tetapi kerangka pengoptimalan menghasilkan algoritma berbasis buffer dan fungsi baru yang memetakan level buffer ke bitrate yang diinginkan. Dalam buku ini juga membuktikan bahwa algoritme menyediakan utilitas yang mendekati optimal.

Membawa algoritma teoretis ke produksi

Tantangan ini adalah merancang algoritma ABR yang berprinsip memerlukan pemodelan yang akurat dari beberapa komponen sistem seperti jaringan dan pemutar video. Namun, model teoritis umumnya tidak dapat menangkap semua kompleksitas sistem produksi. Dengan demikian, menerapkan pekerjaan teoretis dalam produksi hadir dengan serangkaian tantangan baru. Pertimbangan praktis untuk streaming video mencakup kapasitas buffer yang terbatas, perilaku pengguna saat mengontrol pemutar video, dan perubahan kondisi jaringan yang tiba-tiba. Dalam buku ini merancang dua algoritme ABR baru yang menambah algoritme BOLA yang diturunkan secara teoritis untuk bekerja di lingkungan produksi. Algoritme pertama yang disebut BOLA-E memperkenalkan konsep buffer placeholder virtual yang membantu algoritme mengatasi keterbatasan praktis buffer video. Algoritme kedua yang disebut DYNAMIC mendeteksi ketika batasan praktis memengaruhi BOLA dan secara dinamis beralih antara BOLA dan algoritma berbasis throughput dasar yang sesuai. Penerapan algoritme sekarang menjadi bagian dari pemutar referensi resmi DASH dash.js dan sedang digunakan oleh penyedia video di lingkungan produksi.

Protokol transport untuk pengiriman video

Sistem streaming video awal menggunakan Internet sebagai sistem upaya terbaik tanpa transportasi yang andal. Di sisi lain, penyedia video modern menggunakan HAS sebagai metode pengiriman video default, mengirimkan video melalui HTTP melalui TCP atau QUIC. Meskipun ini mengeksploitasi sumber daya besar yang dimiliki oleh jaringan pengiriman konten (CDN) untuk mengirimkan konten HTTP, pemulihan kehilangan paket TCP atau QUIC dapat meningkatkan latensi yang pada gilirannya dapat meningkatkan buffer ulang.

Dalam buku ini merumuskan modifikasi pada protokol QUIC untuk memungkinkan campuran pengiriman yang andal dan tidak andal, mengirimkan header dan bingkai kunci dengan andal, dan mengirimkan bagian non-header dari bingkai non-kunci secara tidak andal. Pendekatan yang disebut VOXEL menghadirkan beberapa kekuatan sistem streaming video awal sambil memungkinkan integrasi yang mudah dengan infrastruktur HTTP yang ada. Faktanya, VOXEL dapat diterapkan secara bertahap dan dapat dioperasikan dengan streaming video HTTP tradisional. Kontribusi utama untuk VOXEL dalam disertasi ini melibatkan augmentasi BOLA-E untuk mengeksploitasi kemampuan yang disediakan oleh VOXEL untuk mengurangi rebuffering dan meningkatkan QoE.

Kerangka kerja simulasi untuk algoritma adaptasi video 360°

Menyampaikan video ubin 360° tidak dapat mengandalkan algoritme 2-D tradisional. Pertama, pemutar video perlu menggunakan algoritme prediksi tampilan untuk memprediksi

bagian mana dari video yang termasuk dalam viewport. Kedua, algoritme ABR 360° perlu menggunakan informasi dari algoritme prediksi tampilan untuk mengalokasikan lebih banyak bandwidth ke petak yang lebih penting. Hal ini membuat desain algoritma lebih menantang. Selain itu, kebutuhan akan HMD membuat pengujian langsung terhadap algoritme baru menjadi lebih sulit.

Dalam buku ini mengembangkan Sabre360, tempat pengujian simulasi untuk video 360° yang memungkinkan pengembangan algoritme 360° secara cepat. Sabre360 mensimulasikan sesi video 360° menggunakan tiga elemen data sebagai input: metadata video, jejak gerakan headset yang direkam untuk video, dan jejak bandwidth yang direkam sebelumnya. Sabre360 juga menggunakan algoritme prediksi tampilan dan algoritme ABR sebagai input. Setelah simulasi sesi, Sabre360 menyediakan log sesi mendetail yang dapat diuraikan untuk menyediakan metrik QoE guna mengevaluasi algoritme. Sabre360 memungkinkan pengujian yang efisien, sehingga memungkinkan untuk menyetel algoritme sebelum tahap pengujian langsung.

BAB 2
BOLA: ADAPTASI BITRATE NEAR-OPTIMAL UNTUK VIDEO ONLINE

Pendekatan berprinsip pada desain algoritme adaptasi kecepatan bit untuk streaming ABR. Secara khusus, dalam buku ini merumuskan adaptasi kecepatan bit sebagai masalah pemaksimalan utilitas yang menggabungkan dua komponen utama QoE: kecepatan bit rata-rata video yang dialami oleh pengguna dan durasi peristiwa penyangga ulang. Peningkatan rata-rata bitrate meningkatkan utilitas, sedangkan rebuffering menurunkannya. Kekuatan kerangka kerja dalam buku ini adalah bahwa utilitas dapat didefinisikan dengan cara yang sangat umum, katakanlah, bergantung pada konten, penyedia video, atau perangkat pengguna.

Dengan menggunakan pengoptimalan Lyapunov, dalam buku ini memperoleh algoritme adaptasi kecepatan bit online yang disebut BOLA (*Algoritma Lyapunov berbasis Buffer Occupancy*) yang terbukti mencapai utilitas yang berada dalam faktor tambahan dari utilitas maksimum yang mungkin dalam rezim video besar. Sementara banyak algoritme adaptasi kecepatan bit telah diusulkan dan diimplementasikan dalam pemutar video, algoritme adalah yang pertama memberikan jaminan teoretis pada utilitas yang dicapai. Selanjutnya, BOLA menyediakan kenop eksplisit bagi penyedia video untuk mengatur kepentingan relatif dari kualitas video yang tinggi dalam kaitannya dengan kemungkinan rebuffering.

Meskipun bukan merupakan bagian eksplisit dari kerangka pengoptimalan Lyapunov, dalam buku ini juga menunjukkan bagaimana BOLA dapat diadaptasi untuk menghindari peralihan kecepatan bit yang sering selama pemutaran video. Sakelar bitrate bisa dibilang tidak terlalu mengganggu dibandingkan dengan rebuffering, tetapi masih menjadi perhatian bagi penyedia video dan pengguna jika sakelar seperti itu terlalu sering terjadi.

Sebagian besar algoritme yang diterapkan dalam praktiknya menggunakan pendekatan berbasis throughput di mana bandwidth yang tersedia antara server dan pemutar video diprediksi dan nilai prediksi digunakan untuk menentukan bitrate segmen berikutnya yang akan diunduh. Pendekatan pelengkap adalah pendekatan berbasis buffer yang tidak memprediksi bandwidth, tetapi hanya menggunakan jumlah data yang saat ini disimpan dalam buffer pemutar video. Baru-baru ini, ada bukti empiris bahwa pendekatan berbasis buffer memiliki sifat yang diinginkan yang kekurangan pendekatan berbasis bandwidth dan telah diadopsi oleh Netflix. Hasil yang menarik dari pekerjaan dalam buku ini adalah bahwa algoritme optimal dalam kerangka pemaksimalan utilitas dalam buku ini hanya memerlukan pengetahuan tentang jumlah data dalam buffer dan tidak ada perkiraan bandwidth yang tersedia. Dengan demikian, pekerjaan dalam buku ini memberikan pembenaran teoretis pertama mengapa algoritma berbasis buffer bekerja dengan baik dalam praktik dan menambahkan wawasan baru ke perdebatan yang sedang berlangsung dalam streaming video

dan komunitas standar DASH tentang kemanjuran relatif dari dua pendekatan. Selanjutnya, karena algoritma BOLA berbasis buffer, ini menghindari overhead prediksi bandwidth yang lebih kompleks yang ada dalam implementasi pemutar video saat ini dan lebih stabil di bawah fluktuasi bandwidth. Perhatikan bahwa hasil menyiratkan bahwa tingkat buffer adalah statistik yang cukup yang secara tidak langsung memberikan semua informasi tentang variasi bandwidth sebelumnya yang diperlukan untuk memilih bitrate berikutnya.

2.1 MODEL SISTEM

Model sistem dengan cermat menangkap cara kerja streaming ABR di Internet saat ini. Dalam buku ini mempertimbangkan pemutar video yang mengunduh file video dari server melalui Internet dan memutarnya kembali ke pengguna. File video disegmentasi menjadi segmen-segmen yang diunduh secara berurutan. Bandwidth yang tersedia antara server dan pemutar bervariasi dari waktu ke waktu. Hal ini dapat terjadi karena alasan seperti kemacetan jaringan dan wireless fading antara lain. Pengalaman menonton pengguna ditentukan oleh kualitas video yang dikuantifikasi oleh bitrate segmen yang diputar dan karakteristik pemutaran seperti rebuffering. Tujuan pemain adalah untuk memaksimalkan utilitas yang terkait dengan pengalaman menonton pengguna sambil beradaptasi dengan perubahan waktu yang bervariasi (dan mungkin tidak dapat diprediksi) dalam bandwidth yang tersedia.

Model Video: File video disegmentasi menjadi N segmen yang diindeks sebagai $\{1, 2, \ldots, N\}$ di mana setiap segmen mewakili p detik dari video. Di server, setiap segmen tersedia dalam M bitrate berbeda di mana segmen yang dikodekan pada bitrate yang lebih tinggi memiliki ukuran bit yang lebih besar dan pemutarannya memberikan pengalaman pengguna yang lebih baik dan utilitas yang lebih tinggi. Misalkan ukuran (dalam bit) dari setiap 1 segmen yang dikodekan pada indeks kecepatan bit m adalah bit Sm dan misalkan utilitas yang diperoleh pengguna dari melihatnya diberikan oleh um di mana m $\in \{1, 2, \ldots, M\}$. WLOG, biarkan bitrate segmen tidak menurun dalam indeks m. Kemudian, berikut ini berlaku.

$$v_1 \leq v_2 \leq \ldots \leq v_M \iff S_1 \leq S_2 \leq \ldots \leq S_M.$$

Perhatikan bahwa kecepatan pengkodean sebenarnya untuk indeks kecepatan bit m diberikan oleh Sm/p bit/detik. Pemutar video mengunduh segmen berurutan dari file video dari server dan memutar ulang segmen yang diunduh ke pengguna. Setiap segmen harus diunduh secara keseluruhan sebelum dapat diputar ulang. Penelitian ini berasumsi bahwa pemain mengirimkan permintaan ke server untuk mengunduh satu segmen dalam satu waktu. Selain itu, segmen diunduh dalam urutan yang sama saat diputar ulang. Pemutar video memiliki buffer terbatas dengan ukuran segmen 2 Q_{max} untuk menyimpan segmen yang telah diunduh tetapi belum diputar ulang. Mengukur buffer dalam segmen sama dengan mengukurnya dalam hitungan detik karena durasi segmen p adalah tetap. Jika buffer penuh, pemutar tidak dapat mendownload segmen baru dan menunggu untuk jangka waktu tetap yang diberikan oleh Δ detik sebelum mencoba mendownload segmen baru. Segmen yang

diunduh sepenuhnya diputar ulang dengan laju tetap $1/p$ segmen/detik tanpa pemalasan apa pun.

Saat mengirim permintaan unduhan untuk segmen baru, pemutar juga menentukan bitrate yang diinginkan untuk segmen tersebut. Hal ini memungkinkan pemutar untuk menukar kualitas video secara keseluruhan dengan kemungkinan buffering ulang yang terjadi saat tidak ada segmen dalam buffer untuk pemutaran. Perhatikan bahwa meskipun setiap segmen memiliki waktu pemutaran tetap p detik, ukuran segmen (dalam bit) dapat berbeda bergantung pada kecepatan bitnya. Dengan demikian, pilihan bitrate untuk suatu segmen memengaruhi waktu pengunduhannya.

Model Jaringan: Bandwidth yang tersedia (dalam bit/detik) antara server dan pemutar diasumsikan bervariasi secara terus-menerus dalam waktu sesuai dengan proses acak stasioner $\omega(t)$. Penulis tidak membuat asumsi tentang mengetahui sifat statistik atau distribusi probabilitas $\omega(t)$ kecuali bahwa ia memiliki momen pertama dan kedua berhingga serta momen invers kedua berhingga. Misalkan pemain mulai mendownload segmen indeks bitrate m pada waktu t. Maka waktu t' saat pengunduhan selesai memenuhi yang berikut:

$$S_m = \int_t^{t'} \omega(\tau) d\tau$$

Biarkan

$$\mathbb{E}\{\omega(t)\} = \omega_{\text{avg}}.$$

Kemudian,

$$\mathbb{E}\{t' - t\} = S_m/\omega_{\text{avg}}.$$

2.2 PERANCANGAN SISTEM

Penulis mempertimbangkan dua metrik kinerja utama yang memengaruhi keseluruhan QoE pengguna: (1) kualitas pemutaran rata-rata waktu yang merupakan fungsi dari bitrate segmen yang dilihat oleh pengguna dan (2) fraksi waktu yang dihabiskan untuk tidak melakukan buffer ulang. Untuk memformalkan metrik ini, penulis mempertimbangkan representasi slot waktu dari model sistem penulis. Garis waktu dibagi menjadi slot berturut-turut yang tidak tumpang tindih dengan panjang variabel dan diindeks oleh $k \in \{1, 2, \ldots\}$. Slot k dimulai pada waktu tk dan berdurasi $Tk = tk + 1 - tk$ detik. Penulis berasumsi bahwa $t1 = 0$. Di awal setiap slot, pemutar video membuat keputusan kontrol apakah harus mulai mengunduh segmen baru, dan jika ya, kecepatan bitnya. Jika keputusan pengunduhan dibuat, maka permintaan dikirim ke server dan pengunduhan segera dimulai. Pengunduhan ini memakan waktu Tk detik dan selesai pada akhir slot k. Perhatikan bahwa Tk adalah variabel acak yang nilai aktualnya bergantung pada realisasi proses $\omega(t)$ serta pilihan bitrate segmen. Jika pemain memutuskan untuk tidak mengunduh segmen baru di slot k (misalnya, saat buffer penuh), maka slot ini berlangsung selama Δ detik.

Buku ini mendefinisikan variabel indikator berikut untuk setiap slot k:

$$a_m(t_k) = \begin{cases} 1 & \text{if the player downloads a segment} \\ & \text{of bitrate index } m \text{ in slot } k, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Maka, untuk semua k, kita harus memiliki $\sum_{m=1}^{M} a_m(t_k) \leq 1$ Selain itu, ketika $\sum_{m=1}^{M} a_m(t_k) = 0$ maka tidak ada segmen yang diunduh. Nyatakan level buffer (diukur dalam jumlah segmen) di awal slot k dengan $Q(t_k)$. Dinamika antrian ini dapat dinyatakan dengan menggunakan persamaan berikut:

$$Q(t_{k+1}) = \max[Q(t_k) - \frac{T_k}{p}, 0] + \sum_{m=1}^{M} a_m(t_k)$$

Di sini, nilai kedatangan ke antrian ini di slot k diberikan oleh $\sum_{m=1}^{M} a_m(t_k)$ yaitu 1 jika keputusan pengunduhan dibuat di slot k dan 0 sebaliknya. Nilai keberangkatan adalah T_k/p yang mewakili jumlah total segmen (termasuk segmen fraksional) yang dapat meninggalkan buffer di slot k. Perhatikan bahwa nilai sebenarnya dari T_k terungkap pada akhir slot k. Perhatikan juga bahwa segmen yang diunduh di slot k hanya tersedia untuk diputar dari slot berikutnya. Penulis berasumsi bahwa level buffer diinisialisasi ke 0, yaitu, $Q(t_1) = 0$.

Biarkan K_N menunjukkan indeks slot di mana segmen ke-N (yaitu, terakhir) diunduh. Juga, tunjukkan waktu di mana pemain selesai memainkan segmen terakhir dengan T_{end}. Kemudian metrik kinerja pertama yang menarik adalah waktu rata-rata yang diharapkan utilitas pemutaran $\bar{v}N$ yang didefinisikan sebagai

$$\overline{v}_N \triangleq \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)v_m\right\}}{\mathbb{E}\{T_{\text{end}}\}}$$

di mana pembilang menunjukkan utilitas total yang diharapkan di semua segmen N. Perhatikan bahwa segmen hanya dapat diputar ulang setelah diunduh seluruhnya. Dengan demikian, T_{end} lebih besar dari waktu selesai pengunduhan segmen terakhir, yaitu $T_{end} > t_{K_N} + T_{K_N}$

Metrik kinerja kedua yang menarik adalah fraksi waktu yang diharapkan $\bar{S}N$ yang dihabiskan tidak melakukan rebuffering dan dapat diartikan sebagai ukuran rata-rata "kelancaran" pemutaran. Ini dapat dihitung dengan mengamati bahwa waktu pemutaran sebenarnya untuk semua N segmen adalah Np detik. Dengan demikian, kelancaran pemutaran yang diharapkan $\bar{S}N$ diberikan oleh

$$\overline{s}_N \triangleq \frac{Np}{\mathbb{E}\{T_{\text{end}}\}} = \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)p\right\}}{\mathbb{E}\{T_{\text{end}}\}}$$

dimana pada langkah terakhir kita menggunakan relasi bahwa $N_p = \sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)p$ Perhatikan bahwa $T_{end} \geq Np$ (karena paling banyak satu segmen dapat diputar kapan saja), sehingga $\bar{S}_N \leq 1$

Tujuan Desain

Penulis ingin merancang sebuah algoritma kontrol yang memaksimalkan utilitas gabungan $\bar{v}_N + \gamma\bar{S}_N$ tunduk pada batasan bahwa $Q_{(t_k)} \geq Q_{max}$ untuk semua k. Di sini, $\gamma > 0$ adalah parameter bobot input untuk memprioritaskan utilitas pemutaran versus kelancaran pemutaran.

Masalah ini dapat diformulasikan sebagai masalah optimasi stokastik dengan tujuan rata-rata waktu di atas cakrawala yang terbatas dan pendekatan berbasis pemrograman dinamis (DP) dapat digunakan untuk menyelesaikannya. Namun, metode berbasis DP tradisional memiliki dua kelemahan utama. Pertama, mereka membutuhkan pengetahuan tentang distribusi proses $\omega(t)$ yang mungkin sulit diperoleh. Kedua, meskipun pengetahuan tersebut tersedia, DP yang dihasilkan dapat memiliki ruang keadaan yang sangat besar. Ini karena ruang status untuk masalah ini di bawah formulasi DP tidak hanya terdiri dari indeks slot waktu k dan nilai tk, tetapi juga ukuran buffer $Q(t_k)$. Selanjutnya, diskritisasi yang sesuai dari proses $\omega(t)$ akan diperlukan untuk mendapatkan solusi yang dapat ditelusuri.

Untuk mengatasi tantangan di atas terkait dengan metode berbasis DP tradisional, penulis mengambil pendekatan alternatif dalam bab ini. Pertama, penulis mempertimbangkan masalah adaptasi bitrate dalam rezim pembatasan ketika ukuran video menjadi besar, yaitu $N \to \infty$. Kedua, penulis mengganti batasan buffer terbatas dengan batasan stabilitas laju (tepatnya di bagian selanjutnya). Alasan untuk membuat asumsi-asumsi ini adalah karena menghasilkan penyederhanaan terhadap masalah asli seperti yang dibahas pada bagian selanjutnya. Hal ini memungkinkan penulis mengembangkan algoritme adaptasi bitrate yang tidak memerlukan pengetahuan apa pun tentang distribusi $\omega(t)$, namun menawarkan jaminan kinerja teoretis yang dapat dibuktikan dalam rezim ukuran video besar sembari memenuhi batasan buffer terbatas. Dengan sedikit modifikasi, algoritme ini juga dapat digunakan untuk video berukuran terbatas dan menawarkan kinerja yang mendekati optimal dalam eksperimen penulis.

Relaksasi Masalah

Pertimbangkan masalah adaptasi bitrate dalam rezim pembatasan ketika ukuran video menjadi besar, yaitu, N → ∞. Kemudian, metrik $\bar{v}_N$ dan $\bar{s}_N$ dapat dinyatakan sebagai

$$\begin{aligned}
\overline{v} \triangleq \lim_{N\to\infty} \overline{v}_N &= \lim_{N\to\infty} \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)v_m\right\}}{\mathbb{E}\left\{T_{\text{end}}\right\}} \\
&= \frac{\lim_{K_N\to\infty}\frac{1}{K_N}\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)v_m\right\}}{\lim_{K_N\to\infty}\frac{1}{K_N}\mathbb{E}\left\{\sum_{k=1}^{K_N} T_k\right\}} \\
\overline{s} \triangleq \lim_{N\to\infty} \overline{s}_N &= \lim_{N\to\infty} \frac{\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)p\right\}}{\mathbb{E}\left\{T_{\text{end}}\right\}} \\
&= \frac{\lim_{K_N\to\infty}\frac{1}{K_N}\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m(t_k)p\right\}}{\lim_{K_N\to\infty}\frac{1}{K_N}\mathbb{E}\left\{\sum_{k=1}^{K_N} T_k\right\}}
\end{aligned}$$

Ini mengikuti dengan mencatat bahwa perbedaan antara waktu selesai pemutaran total yang diharapkan $\mathrm{E}\{T_{\mathrm{end}}\}$ dan waktu penyelesaian unduhan total yang diharapkan $\mathrm{E}\{\sum_{k=1}^{K_N} T_k\}$

dibatasi atas oleh nilai terbatas karena Q_{max} terbatas. Secara khusus, batas atas ini diberikan oleh $Q_{max}p$. Oleh karena itu, alih-alih mempertimbangkan total waktu selesai pemutaran, penulis dapat mempertimbangkan total waktu selesai pengunduhan sebagai tujuan saat ukuran video menjadi besar.

Selanjutnya, ganti batasan finite buffer dengan batasan stabilitas laju. Batasan ini hanya mensyaratkan bahwa laju kedatangan rata-rata waktu ke buffer tidak boleh melebihi laju pemutaran rata-rata waktu. Ini setara dengan mensyaratkan itu

$$\lim_{K_N\to\infty}\frac{1}{K_N}\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M}a_m(t_k)p\right\}\leq\lim_{K_N\to\infty}\frac{1}{K_N}\mathbb{E}\left\{\sum_{k=1}^{K_N}T_k\right\}$$

Batasan stabilitas laju adalah pelonggaran batasan buffer terbatas karena setiap kebijakan yang memastikan buffer terbatas selalu stabil tetapi tidak sebaliknya. Oleh karena itu, di bawah relaksasi ini, utilitas rata-rata waktu yang optimal tidak boleh lebih kecil dari utilitas rata-rata waktu yang optimal dengan batasan penyangga yang terbatas.

Dengan relaksasi ini, tujuan kinerja penulis untuk masalah adaptasi bitrate adalah untuk memaksimalkan utilitas gabungan $\bar{v} + \gamma_{\bar{s}}$ yang tunduk pada batasan stabilitas laju. Mari kita tunjukkan utilitas rata-rata waktu yang optimal untuk rumus ini dengan $v* + \gamma s*$. Masalah ini sesuai dengan kerangka optimisasi Lyapunov untuk sistem pembaruan. Secara khusus, kerangka kerja ini memperluas teknik optimisasi Lyapunov asli ke sistem dengan kerangka pembaharuan panjang variabel dan menunjukkan bahwa meminimalkan "drift-plus-penalti" rasio atas setiap frame menghasilkan algoritma kontrol yang optimal. Penulis merujuk ke untuk detail tentang metode ini. Dalam konteks masalah adaptasi kecepatan bit penulis, slot panjang variabel mewakili bingkai pembaruan. Karakterisasi berikut dapat dibuat tentang optimalitas $i.i.d.$ algoritma.

Lemma 1 Untuk masalah adaptasi bitrate dalam rezim pembatas ketika ukuran video menjadi besar, yaitu, $N \to \infty$, ada algoritma stasioner buffer-state-independent yang membuat i.i.d. mengontrol keputusan di setiap slot dan memenuhi batasan stabilitas laju sambil mencapai utilitas rata-rata waktu tidak lebih kecil dari $v* + \gamma s*$.

Ini mengikuti dari Lemma 1 dan menggunakan fakta bahwa ekspektasi bersyarat dan momen kedua bersyarat dari panjang bingkai dan utilitas dibatasi di bawah algoritma apa pun. Bukti lengkap dihilangkan untuk singkatnya. Perhatikan bahwa algoritma stasioner buffer-state-independent seperti itu belum tentu layak untuk sistem buffer terbatas penulis. Selanjutnya, menghitungnya secara eksplisit membutuhkan pengetahuan tentang distribusi $\omega(t)$. Namun, alih-alih menghitung kebijakan ini secara eksplisit, penulis akan menggunakan keberadaan dan karakterisasi per Lemma 1 untuk merancang algoritme kontrol online menggunakan teknik pengoptimalan Lyapunov pada bingkai pembaruan.Pada bagian berikutnya, penulis akan menyajikan algoritme ini dan menunjukkan bahwa algoritme ini memenuhi batasan buffer terbatas sambil mencapai utilitas rata-rata waktu yang berada

dalam $O(1/Qmax)$ dari $\upsilon * + \gamma s *$ tanpa memerlukan pengetahuan apa pun tentang distribusi $\omega(T)$.

2.3 BOLA: ALGORITMA KONTROL ONLINE

Penulis pertama-tama memberikan intuisi tingkat tinggi dari optimasi Lyapunov melalui teknik pembaruan. Teknik ini mengubah masalah pengoptimalan metrik rata-rata waktu dalam serangkaian masalah pengoptimalan per slot. Masalah yang harus dipecahkan di setiap slot melibatkan meminimalkan rasio nilai drift-plus-penalti yang diharapkan di slot tersebut dengan panjang slot yang diharapkan. Seperti ditunjukkan pada Lampiran, hal ini dapat dilakukan tanpa memerlukan pengetahuan tentang distribusi $\omega(t)$. Suku drift terdiri dari $\mathrm{E}\left\{\left(Q\left(t_{k+1}\right)\right)^2 - Q(t_k)^2 : 2 \mid Q(t_k)\right\}$ dan memenuhi batasan stabilitas laju. Istilah penalti terdiri dari utilitas pemutaran dan kelancaran pemutaran yang diterima di slot itu. Penulis menjaga utilitas dan kehalusan sebagai istilah terpisah meskipun dapat dilipat menjadi satu metrik. Hal ini memungkinkan penulis menyesuaikan kepentingan relatif untuk meningkatkan bitrate video dan mengurangi buffering ulang tanpa mengubah algoritme. Algoritme menggunakan parameter kontrol $V > 0$ untuk memungkinkan pertukaran antara ukuran buffer dan tujuan kinerja.

Penulis sekarang menyajikan algoritma. Di setiap slot k, diberi level buffer $Q(t_k)$ di awal slot, algoritme penulis membuat keputusan kontrol dengan menyelesaikan masalah optimisasi deterministik berikut.

$$\rho(t_k, \mathbf{a}(t_k)) = \begin{cases} 0 & \text{if } \sum_{m=1}^{M} a_m(t_k) = 0, \\ \dfrac{\sum_{m=1}^{M} a_m(t_k)\left(V v_m + V\gamma p - Q(t_k)\right)}{\sum_{m=1}^{M} a_m(t_k) S_m} & \\ & \text{otherwise.} \end{cases}$$

Kemudian tentukan $\mathbf{a}(t_k)$ dengan menyelesaikan masalah optimisasi:

Maksimalkan : $\rho(t_k, \boldsymbol{a}(t_k))$

Subjeknya pada :

$$\sum_{m=1}^{M} a_m(t_k) \leq 1, a_m(t_k) \in \{0, 1\}$$

Kendala dari masalah ini menghasilkan struktur solusi yang sangat sederhana. Secara khusus, solusi optimal diberikan oleh:

1. Jika $Q(t_k) > V(v_m + \gamma\rho)$ untuk semua $m \in \{1, 2, \ldots, M\}$, maka opsi tanpa unduhan dipilih, yaitu, am(tk) = 0 untuk semua m. Perhatikan bahwa dalam hal ini $T_k = \Delta$.
2. Selain itu, solusi optimal adalah mengunduh segmen berikutnya pada indeks bitrate $m*$ di mana $m*$ adalah indeks yang memaksimalkan rasio $(\mathrm{V}v_m + \mathrm{V}_{\gamma\rho} - Q(t_k))/S_m$ di antara semua m yang rasionya positif.

Perhatikan bahwa menyelesaikan soal ini tidak membutuhkan pengetahuan tentang proses $\omega(t)$. Selanjutnya, solusi optimal hanya bergantung pada level buffer $Q(t_k)$. Itu sebabnya penelitian ini menyebut algoritme penulis BOLA: Buffer Occupancy berdasarkan Algoritma Lyapunov. Properti BOLA ini harus dikontraskan dengan strategi berbasis prediksi bandwidth yang baru-baru ini diusulkan untuk masalah ini yang membutuhkan prediksi eksplisit dari bandwidth yang tersedia untuk keputusan kontrol.

Teorema berikut mencirikan jaminan kinerja teoretis yang disediakan oleh BOLA.

Teorema 2 Misalkan BOLA seperti yang didefinisikan oleh dipersamaan diatas dan diimplementasikan di setiap slot menggunakan parameter kontrol $0 < \mathrm{V} \leq \frac{Q_{max}-1}{vM+\gamma\rho}$. Asumsikan $Q(0) = 0$.

Kemudian, penangguhan berikutnya.

1. Backlog antrian memenuhi $Q(t_k) \leq V(v_m + \gamma\rho) + 1$ untuk semua slot k. Lebih lanjut, hunian buffer di segmen tidak pernah melebihi Q_{max}.
2. Utilitas rata-rata waktu yang dicapai oleh BOLA memuaskan

$$\overline{v}^{\mathrm{BOLA}} + \gamma\overline{s}^{\mathrm{BOLA}} \geq v^* + \gamma s^* - \frac{p^2 + \Psi}{2p^2 V}$$

di mana Ψ adalah batas atas $\mathrm{E}\{T_k^2\}$ di bawah algoritma kontrol apa pun dan dianggap terbatas.

Bukti: pertama kali menunjukkan bagian 1 menggunakan induksi. Perhatikan bahwa ikatan $Q(t_k) \leq V(v_m + \gamma\rho) + 1$ berlaku untuk $k = 1$ karena $Q(t_1) = Q(0) = 0$. Sekarang misalkan berlaku untuk beberapa k. Buku ini akan menunjukkan bahwa itu juga berlaku untuk $k + 1$. Penulis memiliki dua kasus.

Kasus 1: $Q(t_k) \leq V(v_m + \gamma\rho)$

Dari persamaan antrian, dapat disimpulkan bahwa $Q(t_k)$ maksimum yang dapat meningkat di slot k adalah sebesar 1. Hal ini mengimplikasikan bahwa $Q(t_k + 1) \leq V(v_m + \gamma\rho) + 1$.

Kasus 2: $V(v_m + \gamma\rho) < Q(t_k) \leq (u_m + \gamma\rho) + 1$

Kita memiliki $Q(t_k) > V(v_m + \gamma\rho)$ untuk semua m ∈ $\{1, 2, \ldots, M\}$. Ini mengikuti dari struktur solusi optimal bahwa BOLA akan memilih opsi tanpa pengunduhan dalam kasus ini. Akibatnya, $Q(t_k)$ tidak dapat meningkat dan kita mendapatkan bahwa $Q(t_k + 1) \leq V(v_m + \gamma\rho) + 1$.

$Q(t_k)$ menunjukkan jumlah segmen dalam buffer. Ini bisa paling banyak Q_{max} menggunakan relasi

$$V \leq \frac{Q_{max} - 1}{vM + \gamma\rho}$$

Buku ini menunjukkan batas dalam menggunakan teknik optimalisasi Lyapounov pada frame ukuran variabel. Pertama-tama kita mendefinisikan fungsi Lyapunov $LQ(t_k)$ sebagai

$$L(Q(t_k)) = \frac{1}{2}Q^2(t_k)$$

dan tentukan per-slot conditional Lyapunov drift D(t_k) sebagai

$$D(t_k) \triangleq \mathbb{E}\left\{L(Q(t_{k+1})) - L(Q(t_k))|Q(t_k)\right\}$$

menggunakan persamaan antrian, untuk mengikat $D(t_k)$ Penulis mempertimbangkan dua kasus untuk: $LQ(t_k) \leq T_k/p$ dan $Q(t_k) > T_k/p$. Dalam kasus pertama yang kita miliki

$$D(t_k) = \mathbb{E}\left\{\frac{1}{2}\left(\sum_{m=1}^{M} a_m(t_k)\right)^2 - \frac{1}{2}Q^2(t_k)|Q(t_k)\right\}$$

Dalam kasus kedua kita punya

$$D(t_k) = \mathbb{E}\left\{\frac{1}{2}\left(\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k)\right)^2 - Q(t_k)\left(\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k)\right)|Q(t_k)\right\}$$

Dalam kedua kasus, $D(t_k)$ dibatasi dengan

$$D(t_k) \leq \frac{p^2 + \Psi}{2p^2} - Q(t_k)\mathbb{E}\left\{\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k)|Q(t_k)\right\}$$

di mana Ψ adalah batas atas pada $\mathbf{E}\{T_k^2\}$ di bawah algoritma kontrol apa pun dan dianggap terbatas.

Mengikuti metodologi teknik pengoptimalan Lyapunov, penulis mengurangi V × istilah hadiah dari kedua sisi di atas untuk mendapatkan

$$\begin{aligned} D(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m(t_k)(v_m + \gamma p)|Q(t_k)\right\} \\ \leq \frac{p^2 + \Psi}{2p^2} - Q(t_k)\mathbb{E}\left\{\frac{T_k}{p} - \sum_{m=1}^{M} a_m(t_k)|Q(t_k)\right\} \\ - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m(t_k)(v_m + \gamma p)|Q(t_k)\right\} \end{aligned}$$

Mari kita tunjukkan keputusan kontrol (dan panjang slot yang dihasilkan) di bawah algoritma kontrol penulis oleh BOLA superskrip sedangkan yang di bawah kebijakan stasioner Lemma 1 oleh STAT. Karena BOLA dengan rakus memaksimalkan bingkai, itu memastikannya

$$\begin{aligned} \mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(Q(t_k) - V(v_m + \gamma p))|Q(t_k)\right\} \\ \leq \eta \times \mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{STAT}}(t_k)(Q(t_k) - V(v_m + \gamma p))|Q(t_k)\right\} \end{aligned}$$

Di mana $\eta = \frac{\mathbf{E}\{T_k^{BOLA} \mid Q(t_k)\}}{\mathbf{E}\{T_k^{STAT} \mid Q(t_k)\}}$ untuk melihatnya, bandingkan rasio di sisi kiri di atas dengan tujuan sambil mencatat bahwa kita dapat menyatakan penyebutnya sebagai

$\mathbf{E}\{T_k^{BOLA} \mid Q(t_k)\} = \frac{(\sum_{m-1}^{M} a_m^{BOLA}(t_k) S_m)}{\omega_{avg}}$. Perlu dicatat bahwa rasio ini dapat diminimalkan tanpa memerlukan pengetahuan tentang ω_{avg}

$$D^{\mathrm{BOLA}}(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(v_m + \gamma p)|Q(t_k)\right\}$$
$$\leq \frac{p^2+\Psi}{2p^2} - Q(t_k)\mathbb{E}\left\{\frac{T_k^{\mathrm{BOLA}}}{p} - \eta\sum_{m=1}^{M} a_m^{\mathrm{STAT}}(t_k)|Q(t_k)\right\}$$
$$- V\eta\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{STAT}}(t_k)(v_m + \gamma p)|Q(t_k)\right\}$$

Mengganti nilai rata-rata waktu untuk kebijakan stasioner yang kita dapatkan

$$D^{\mathrm{BOLA}}(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(v_m + \gamma p)|Q(t_k)\right\}$$
$$\leq \frac{p^2+\Psi}{2p^2} - Q(t_k)\left(\frac{1}{p} - r^{\mathrm{STAT}}\right)\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\}$$
$$- V(v^* + \gamma s^*)\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\}$$

di mana r^{STAT} menunjukkan tingkat kedatangan yang diharapkan di bawah kebijakan stasioner dan tidak dapat melebihi $1/p$ karena tingkat stabil. Jadi kita punya

$$D^{\mathrm{BOLA}}(t_k) - V\mathbb{E}\left\{\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(v_m + \gamma p)|Q(t_k)\right\}$$
$$\leq \frac{p^2+\Psi}{2p^2} - V(v^* + \gamma s^*)\mathbb{E}\left\{T_k^{\mathrm{BOLA}}|Q(t_k)\right\}$$

Mengambil ekspektasi bersyarat dari kedua sisi dan menjumlahkan $k \in \{1, 2, \ldots, KN\}$, kita dapatkan

$$\mathbb{E}\left\{L(Q(t_{K_{N+1}}))\right\} - V\mathbb{E}\left\{\sum_{k=1}^{K_N}\sum_{m=1}^{M} a_m^{\mathrm{BOLA}}(t_k)(v_m + \gamma p)\right\}$$
$$\leq \frac{(p^2+\Psi)K_N}{2p^2} - V(v^* + \gamma s^*)\mathbb{E}\left\{\sum_{k=1}^{K_N} T_k^{\mathrm{BOLA}}\right\}$$

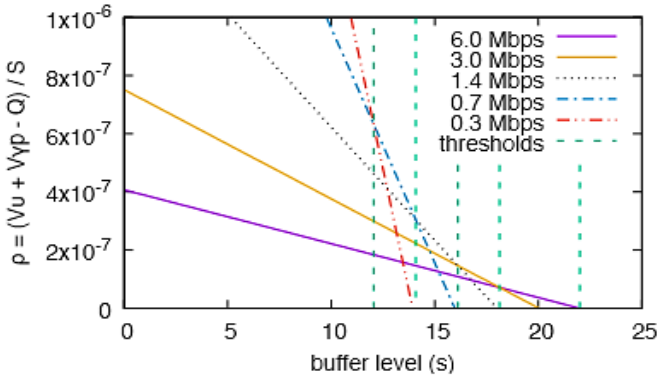
Membagi kedua sisi dengan $VE\{\sum_{k=1}^{K_N} T_k^{BOLA}\}$ dan mengambil batas sebagai $N \to \infty$ menghasilkan batas dalam. Batasan kinerja dalam Teorema 2 menunjukkan tradeoff $[O(1/V), O(V)]$ utilitas dan backlog yang tipikal dari algoritme kontrol berbasis Lyapunov untuk masalah pemaksimalan utilitas serupa. Secara khusus, utilitas rata-rata waktu BOLA berada dalam suku aditif $O(1/V)$ dari utilitas optimal dan celah ini dapat dibuat lebih kecil dengan memilih nilai V yang lebih besar. Namun, nilai layak terbesar dari V dibatasi oleh ukuran buffer dan terdapat hubungan linier di antara keduanya.

Pengertian BOLA Beserta Contoh

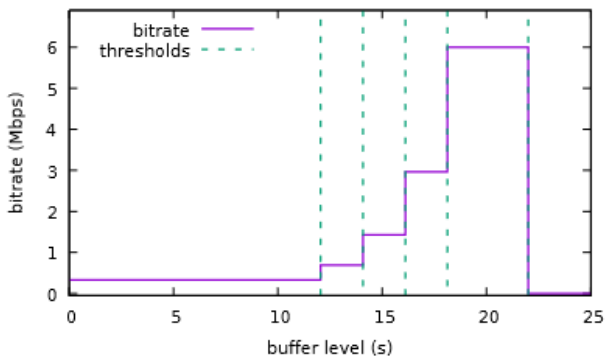
Buku ini menyajikan contoh proses untuk mengilustrasikan cara kerja BOLA. Penulis mengiris video 99 detik menggunakan segmen 3 detik dan menyandikannya pada lima bitrate berbeda. Sementara BOLA hanya membutuhkan utilitas untuk menjadi fungsi yang tidak menurun dari bitrate segmen, wajar untuk mempertimbangkan fungsi utilitas cekung dengan hasil yang semakin berkurang, misalnya, peningkatan 1 Mbps pada bitrate segmen kemungkinan memberikan keuntungan utilitas yang lebih besar bagi pengguna saat itu. peningkatan dari 0,5 Mbps menjadi 1,5 Mbps dibandingkan ketika dari 5 Mbps menjadi 6 Mbps. Pilihan alami untuk contoh kita adalah fungsi utilitas logaritmik: misalkan $v_m = \ln(S_m/S_1)$. Pilih $\gamma = 5.0/p$ dan $V = 0.93$. Bitrate dan utilitas ada di bawah.

Bitrate (Mbps)	0.331	0.688	1.427	2.962	6.000
S (Mb)	0.993	2.064	4.281	8.886	18.00
v	0.000	0.732	1.461	2.192	2.897

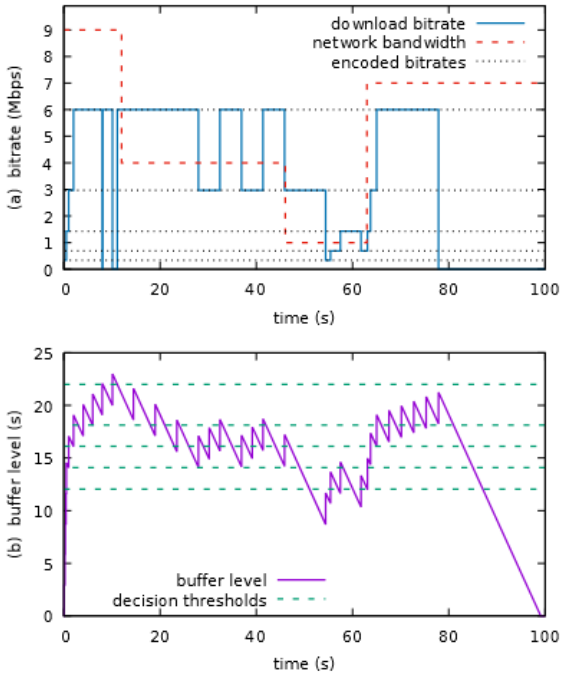
Untuk setiap slot penulis memilih bitrate segmen untuk dimaksimalkan $(Vv_m + V_{\gamma p} - Q)/S_m$ untuk $1 \leq m \leq M$. Gambar 2.1 menunjukkan hubungan antara ekspresi dan level buffer Q untuk m yang berbeda. Persimpangan garis menandai tingkat penyangga yang sesuai dengan ambang batas keputusan. Gambar 2.2 merangkum pilihan bitrate BOLA sebagai fungsi dari level buffer.



Gambar 2.1. Nilai (V u_m + V γp Q)/Sm untuk kecepatan bit yang berbeda bergantung pada tingkat buffer. (γp = 5 dan V = 0,93.) Perhatikan bahwa tingkat buffer adalah Qp detik.



Gambar 2.2. Pilihan bitrate BOLA sebagai fungsi dari level buffer. (γp = 5,V = 0,93.) Perhatikan bahwa tingkat buffer adalah Qp detik.



Gambar 2.3. Contoh unduhan dan pemutaran video menggunakan BOLA. (a) Video dikodekan pada 5 bitrate yang berbeda. Bandwidth jaringan bervariasi dari tinggi ke rendah dan kembali ke tinggi. Bitrate segmen yang diunduh menyesuaikan dengan bandwidth jaringan. (b) Variasi level buffer memicu perubahan bitrate saat melintasi ambang batas.

Gambar 2.3 menunjukkan cara kerja BOLA. Penulis menggunakan profil bandwidth jaringan sintetik seperti yang ditunjukkan pada Gambar 2.3(a). Kita dapat melihat loop umpan balik yang melibatkan bitrate di (a) dan level buffer di (b). BOLA memilih bitrate langsung berdasarkan level buffer menggunakan Gambar 2.2. Bitrate memengaruhi waktu pengunduhan, sehingga secara tidak langsung memengaruhi level buffer di awal slot berikutnya. Terakhir, setelah semua segmen diunduh, pemutar video memutar segmen yang tersisa di buffer.

Sementara penulis memilih fungsi utilitas logaritmik sebagai contoh, penyedia video dapat menggunakan fungsi utilitas apa pun yang memuaskan. Fungsi utilitas mungkin juga mempertimbangkan karakteristik sistem seperti jenis perangkat yang digunakan pemirsa.

γ sesuai dengan seberapa kuat kita ingin menghindari rebuffering. Peningkatan γ menerjemahkan grafik dalam Gambar 2.1 dan 2.2 dibab sebelumnya secara efektif menggeser ambang lebih tinggi tanpa mengubah jarak relatifnya. BOLA karenanya akan mengunduh lebih banyak segmen bitrate rendah untuk mempertahankan tingkat buffer yang lebih besar (dan lebih aman).

Meningkatkan V memperluas grafik pada Gambar. 2.1 dan 2.2 secara horizontal tentang asal. Jika kita memiliki level buffer maksimum Q_{max}, kita ingin menghindari pengunduhan kecuali ada cukup ruang untuk satu segmen penuh pada *buffer*, yaitu kecuali $Q \leq Q_{max} - 1$. Untuk Q_{max} tertentu kita dapat mengatur $V = (Q_{max} - 1)/(\upsilon M + \gamma p)$.

Sementara penulis menunjukkan cara memilih nilai wajar untuk γ dan V , penyedia video lebih terbiasa memilih target tingkat buffer. Alternatifnya, penyedia video dapat memilih γ dan V untuk menyetel otomatis parameter BOLA.

BAB 3
MEMILIH UTILITAS DAN PARAMETER

3.1 IMPLEMENTASI DAN EVALUASI EMPIRIS

Penulis pertama kali mengimplementasikan versi dasar BOLA, bernama BOLA-BASIC. Ingatlah bahwa ketika level buffer penuh, BOLA tidak mengunduh segmen tetapi menunggu Δ detik. Daripada memilih nilai arbitrer untuk Δ, penulis menggunakan menunggu dinamis hingga $Q(t_k) \leq V\ (\upsilon M + \gamma p)$. Ini memiliki efek yang sama dengan memilih Δ yang tetap tetapi sangat kecil, sehingga analisis teoretis masih berlaku. Penulis juga menerapkan versi BOLA lainnya, yaitu BOLA-FINITE, BOLA-O, dan BOLA-U, yang akan dijelaskan nanti di bagian ini.

Tabel 3.1. Bitrate digunakan untuk Video Uji Big Buck Bunny

Bitrate Index *m*	Bitrate Mean	Std Dev (Mbps)	Segmen Mean	Std Dev (Mb)	Utility v = ln(S/S_1)
1	0.230	0.038	0.690	0.113	0.000
2	0.331	0.054	0.993	0.162	0.364
3	0.477	0.096	1.431	0.287	0.729
4	0.688	0.120	2.064	0.360	1.096
5	0.991	0.182	2.973	0.545	1.461
6	1.427	0.275	4.281	0.825	1.825
7	2.056	0.394	6.168	1.182	2.190
8	2.962	0.564	8.886	1.691	2.556
9	5.027	0.891	15.08	2.673	3.084
M=10	6.000	1.078	18.00	3.232	3.261

3.2 METODOLOGI PENGUJIAN

Dalam buku ini mensimulasikan semua versi BOLA menggunakan film Big Buck Bunny. Film berdurasi 10 menit itu dikodekan pada 10 bitrate berbeda dan diiris dalam segmen 3 detik. Meskipun setiap indeks kualitas memiliki kecepatan bit rata-rata tertentu, segmen mungkin memiliki kecepatan bit variabel (VBR) karena sifat film yang berbeda-beda. Penulis mensimulasikan waktu pemutaran lebih lama dari 10 menit dengan mengulang film. Sekali lagi penulis memilih fungsi utilitas logaritmik:

$$v_m = \ln(\frac{S_m}{S_1})$$

Tabel 3.1 menunjukkan mean dan standar deviasi dari bitrate dan ukuran segmen untuk setiap indeks kualitas dan nilai utilitas masing-masing.

Forum Industri DASH memberikan tolok ukur untuk berbagai aspek standar DASH [14]. Tolok ukur mencakup dua belas profil jaringan yang berbeda. Profil 1–6 memiliki bandwidth jaringan mulai dari 1,5 hingga 5 Mbps sedangkan profil 7–12 memiliki bandwidth mulai dari 1

hingga 9 Mbps. Latensi yang berbeda disediakan untuk setiap bandwidth, di mana latensi adalah setengah dari waktu pulang pergi (RTT). Tabel 2.2 menunjukkan karakteristik bandwidth bernomor ganjil. Profil 1 menghabiskan 30 detik pada masing-masing 5, 4, 3, 2, 1,5, 2, 3, dan 4 Mbps, lalu mulai kembali dari atas. Profil bernomor genap mirip dengan profil bernomor ganjil sebelumnya tetapi dimulai pada tahap bandwidth rendah. Misalnya, profil 2 mulai dari 1,5 Mbps.

Selain itu, penulis juga menguji algoritme penulis menggunakan satu set 86 jejak bandwidth seluler 3G yang tersedia untuk umum. Satu jejak dikecualikan karena memiliki bandwidth rata-rata 80 kbps; bitrate video terendah penulis adalah 230 kbps. Karena pelacakan tidak menyertakan pengukuran latensi, penulis menggunakan latensi 50 md sehingga menghasilkan RTT sepanjang 100 md.

3.3 MENGHITUNG BATAS ATAS PADA UTILITAS MAKSIMUM

Untuk mengevaluasi seberapa baik kinerja BOLA pada jejak, penting untuk menurunkan batas atas pada utilitas maksimum yang dapat diperoleh oleh algoritma apa pun pada jejak yang diberikan. Dalam buku ini mendapatkan algoritme optimal offline yang menyediakan utilitas maksimum yang dapat dicapai menggunakan pemrograman dinamis. Penulis mendefinisikan tabel $r(n,t,b)$ yang berisi utilitas maksimum yang mungkin saat penulis mengunduh segmen ke-n dan menyelesaikannya pada waktu t dengan tingkat buffer b. Penulis menginisialisasi tabel dengan $r(0,0,0) = 0$. Biarkan $x(n,t,m)$ menjadi waktu untuk mengunduh segmen ke-n pada indeks kecepatan bit m mulai dari waktu t. Perhatikan bahwa ketergantungan x pada n adalah karena VBR. Penulis mengkuantisasi waktu dengan perincian δ. Meskipun beberapa keakuratan hilang, penulis memastikan hasil akhir masih batas atas dengan membulatkan waktu pengunduhan ke bawah.

$$x_\delta(n,t,m) = \lfloor x(n,t,m)/\delta \rfloor \cdot \delta$$

Buku ini membatasi level buffer pada b_{max}.

$$x'_\delta(n,t,b,m) = \max[x_\delta(n,t,m), b + p - b_{\max}]$$

Biarkan $y(n,t,b,m)$ menjadi waktu buffering.

$$y(n,t,b,m) = \max[x'_\delta(n,t,b,m) - b, 0]$$

Penulis membuat entri untuk $r(n,\cdot,\cdot)$ dari $r(n-1,\cdot,\cdot)$ menggunakan

$$r(n,t,b) = \max_{m,t',b'} \left(r(n-1,t',b') + v_m - \gamma y(n,t',b',m) \right)$$

seperti yang $t = t' + x'_\delta(n,t',b',m)$ dan $b = b' - x'_\delta(n,t',b',m) + y(n,t',b',m) + p$

Algoritma pemrograman dinamis ditunjukkan pada Gambar 2.4 di bab sebelumnya.

3.4 MENGEVALUASI BOLA-BASIC

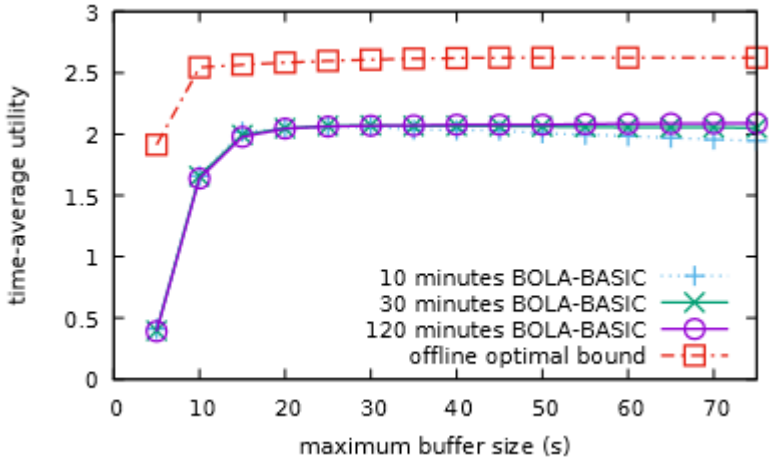
Gambar 2.5 pada bab sebelumnya menunjukkan utilitas rata-rata waktu BOLA-BASIC ketika panjang video adalah 10, 30 dan 120 menit. Penulis menetapkan $\gamma p = 5$ dan memvariasikan V untuk ukuran buffer yang berbeda. Penulis membandingkan utilitas BOLA-BASIC dengan batas optimal luring. Pengoptimalan offline memberikan utilitas yang hampir sama untuk durasi video yang berbeda. BOLA-BASIC hanya memperoleh sekitar 80% dari batas optimal offline. Selain itu, utilitas BOLA-BASIC sedikit berkurang ketika ukuran buffer ditingkatkan karena harus mengunduh lebih banyak segmen bitrate rendah selama startup sebelum dapat mencapai level buffer yang diperlukan untuk beralih ke segmen bitrate lebih tinggi. Hasil penulis menunjukkan bahwa ada ruang untuk meningkatkan BOLA-BASIC yang memotivasi versi penulis berikutnya.

$r(0,t,b) \leftarrow \{0 \text{ for } t = b = 0, -\infty \text{ otherwise}\}$
for n in $[1,N]$ **do**
  initialize $r(n,t,b) \leftarrow -\infty$ for all t,b
  for all (t',b') such that $r(n-1,t',b') > -\infty$ **do**
    for m in $[1,M]$ **do**
      $x \leftarrow$ download time(n,t',m)
      $x_\delta \leftarrow \lfloor x/\delta \rfloor \cdot \delta$
      $x'_\delta \leftarrow \max[x_\delta, b' + p - b_{\max}]$
      $y \leftarrow \max[x'_\delta - b', 0]$
      $t \leftarrow t' + x'_\delta$
      $b \leftarrow b' - x'_\delta + y + p$
      $r' \leftarrow r(n-1,t',b') + v_m - \gamma y$
      $r(n,t,b) \leftarrow \max[r(n,t,b), r']$
    end for
  end for
end for
$r^* \leftarrow \max_{(t,b)} \dfrac{r(N,t,b)}{(t+b)}$

Gambar 3.1. Menghitung Batas Atas Utilitas Optimal Offline

BAB 4
MENGADAPTASI BOLA KE VIDEO BERUKURAN HINGGA

BOLA-BASIC diturunkan dengan asumsi bahwa video tidak terbatas. Oleh karena itu, diperlukan beberapa adaptasi agar BOLA dapat bekerja secara efektif dengan video yang lebih kecil. Termotivasi oleh percobaan awal penulis, penulis menerapkan dua adaptasi BOLA-BASIC untuk mendapatkan versi yang penulis sebut BOLA-FINITE.



Gambar 4.1 . Utilitas rata-rata waktu untuk γp = 5 menggunakan profil 1 untuk BOLA-BASIC.

1) Nilai Dynamic V untuk startup dan wind down: Buffer yang besar memungkinkan BOLA-BASIC bekerja lebih baik tetapi memiliki dua kekurangan. Pertama, butuh waktu lebih lama untuk membuat buffer besar selama startup. Segmen bitrate yang lebih rendah lebih disukai hingga level buffer mencapai kondisi stabil. Kedua, pada beberapa tahap akhir, semua unduhan selesai dan video buffer yang tersisa diputar. Setiap bandwidth yang tersedia selama periode ini tidak digunakan. Mempersingkat periode ini akan menghasilkan lebih sedikit bandwidth tersedia yang tidak terpakai.

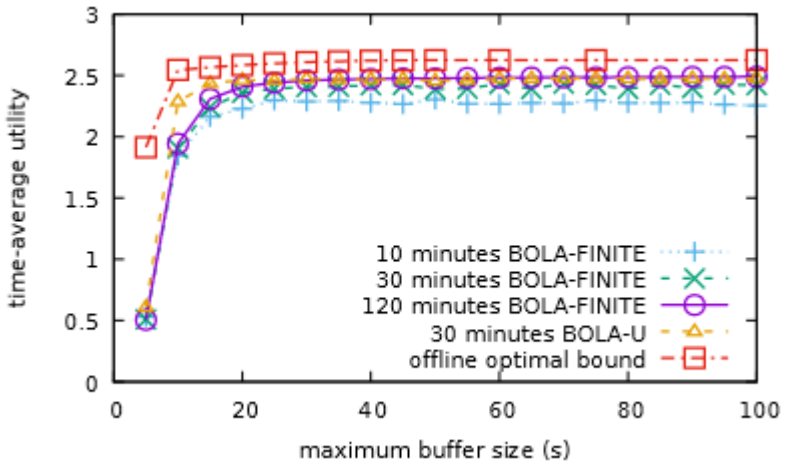
1: **for** n in $[1, N]$ **do**
2: $t \leftarrow \min[\text{playtime from begin}, \text{playtime to end}]$
3: $t' \leftarrow \max[t/2, 3p]$
4: $Q^{\mathrm{D}}_{\max} \leftarrow \min[Q_{\max}, t'/p]$
5: $V^{\mathrm{D}} \leftarrow (Q^{\mathrm{D}}_{\max} - 1)/(v_M + \gamma p)$
6: $m^*[n] \leftarrow \arg\max_m (V^{\mathrm{D}} v_m + V^{\mathrm{D}} \gamma p - Q)/S_m$
7: **if** $m^*[n] > m^*[n-1]$ **then**
8: $r \leftarrow$ bandwidth measured when downloading segment $(n-1)$
9: $m' \leftarrow \max m$ such that $S_m/p \leq \max[r, S_1/p]$
10: **if** $m' \geq m^*[n]$ **then**

11: $m' \leftarrow m^*[n]$
12: **else if** $m' < m^*[n-1]$ **then**
13: $m' \leftarrow m^*[n-1]$
14: **else if** some utility sacrificed for fewer oscillations **then**
15: pause until $(V^{\mathrm{D}} v_{m'} + V^{\mathrm{D}} \gamma p - Q)/S_{m'} \geq (V^{\mathrm{D}} v_{m'+1} + V^{\mathrm{D}} \gamma p - Q)/S_{m'+1}$ ▷ BOLA-O
16: **else**
17: $m' \leftarrow m' + 1$ ▷ BOLA-U
18: **end if**
19: $m^*[n] \leftarrow m'$
20: **end if**
21: pause for $\max[p \cdot (Q - Q^{\mathrm{D}}_{\max} + 1), 0]$
22: download segment n at bitrate index $m^*[n]$, possibly abandoning
23: **end for**

1: **function** SHALLABANDON(m, S^{R}_m)
2: $r_m \leftarrow (V^{\mathrm{D}} v_m + V^{\mathrm{D}} \gamma p - Q)/S^{\mathrm{R}}_m$
3: $r_{m'} \leftarrow (V^{\mathrm{D}} v_{m'} + V^{\mathrm{D}} \gamma p - Q)/S_{m'}$
4: **return** *true* if $r_{m'} > r_m$ for some m' subject to $1 \leq m' < m$
5: **end function**

Gambar 4.2. Heuristik pengabaian unduhan BOLA-FINITE: m adalah bitrate segmen saat ini dan S^{R}_{m} adalah jumlah bit yang tersisa untuk diunduh di segmen saat ini.

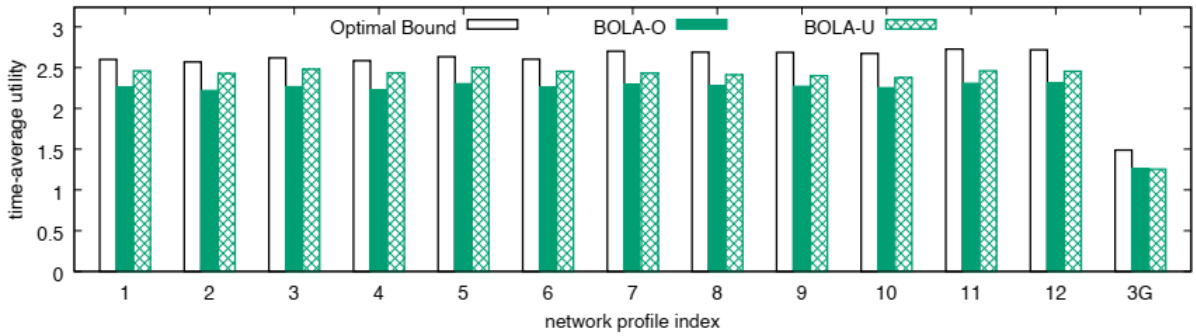
Dalam buku ini mengurangi efek ini dengan memperkenalkan $V\ D$ dinamis yang sesuai dengan QD_{max} ukuran buffer dinamis, ditunjukkan pada baris 2–5 pada Gambar 4.2 BOLA-FINITE tidak mencoba mengisi seluruh buffer terlalu cepat dan tidak mencoba mempertahankan buffer penuh terlalu lama. Penulis masih membutuhkan ukuran buffer minimum $3p$ agar algoritme dapat bekerja secara efektif.



Gambar 4.3. Utilitas rata-rata waktu untuk γp = 5 menggunakan profil 1 untuk BOLA-FINITE dan BOLA-U.

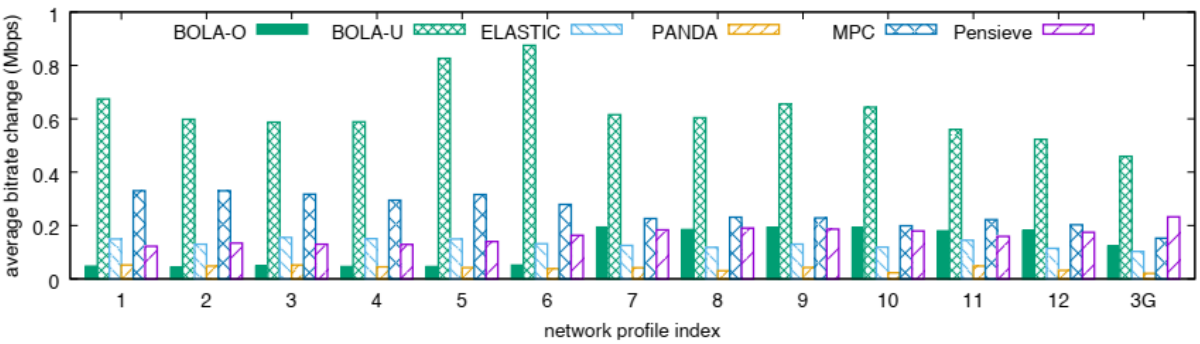
2) Pengabaian unduhan: BOLA-BASIC mengambil keputusan kontrol tepat sebelum mengunduh setiap segmen. Pertimbangkan skenario di mana pemutar mengunduh segmen

bitrate 6 Mbps tinggi dalam kondisi jaringan yang baik. Bandwidth jaringan tiba-tiba turun menjadi 1 Mbps karena pemain baru saja memulai pengunduhan segmen baru. Segmen akan memakan waktu $6p$ detik untuk diunduh, menghabiskan buffer dan mungkin menyebabkan buffering ulang. BOLA-FINITE mengurangi masalah ini dengan memantau kemajuan pengunduhan dan kemungkinan mengabaikan pengunduhan. Gambar 4.4 menunjukkan bagaimana BOLA-FINITE memutuskan apakah akan meninggalkan pengunduhan atau tidak. Jika segmen pada indeks kecepatan bit m sedang diunduh, sisa ukuran SR_m kurang dari S_m.



Gambar 4.4. Utilitas rata-rata waktu BOLA-O dan BOLA-U dengan γp = 5 dan buffer 25 detik memutar video 30 menit untuk profil jaringan uji DASH 1–12 dan pelacakan seluler (3G). Utilitas BOLA berada dalam 84–95% dari utilitas optimal offline.

Segmen dapat ditinggalkan dan diunduh pada beberapa indeks bitrate mJ tunduk pada $1 < m' < m$ ketika $(V^{D_{v_m}} + V^{D_{\gamma p}} - Q)/S_m^R < (V^{D_{v_{m'}}} + V^{D_{\gamma p}} - Q)/S_{m'}$). Gagasan kontrol tetap sama, tetapi bitrate m saat ini memiliki ukuran SR yang lebih kecil karena bagian dari segmen telah diunduh. Gambar 4.5 mengilustrasikan skenario di mana pengabaian dapat membantu. Pada 46 detik, pengunduhan segmen 3 Mbps dimulai. Karena ada penurunan bandwidth pada saat itu, segmen tersebut membutuhkan waktu hampir 9 detik untuk mengunduh. Buffer habis dan BOLA-BASIC beralih ke pengunduhan dengan kecepatan bit 0,3 Mbps. BOLA-FINITE dengan logika pengabaian akan mendeteksi buffer yang menipis dengan cepat dan menghentikan unduhan yang lama, dengan sistem hanya turun ke bitrate unduhan 1,4 dan 0,7 Mbps dalam periode bandwidth rendah.



Gambar 4.5. Perubahan bitrate rata-rata antara segmen yang berdekatan lebih kecil untuk BOLA-O daripada BOLA-U, tetapi beberapa perubahan bitrate diperlukan untuk melacak bandwidth jaringan secara akurat. Dalam percobaan penulis, sebagai rata-rata

di seluruh profil jaringan, ELASTIC dan PANDA melacak bandwidth dengan akurasi yang mirip dengan BOLA-O, sementara MPC dan Pensieve memiliki lebih banyak osilasi.

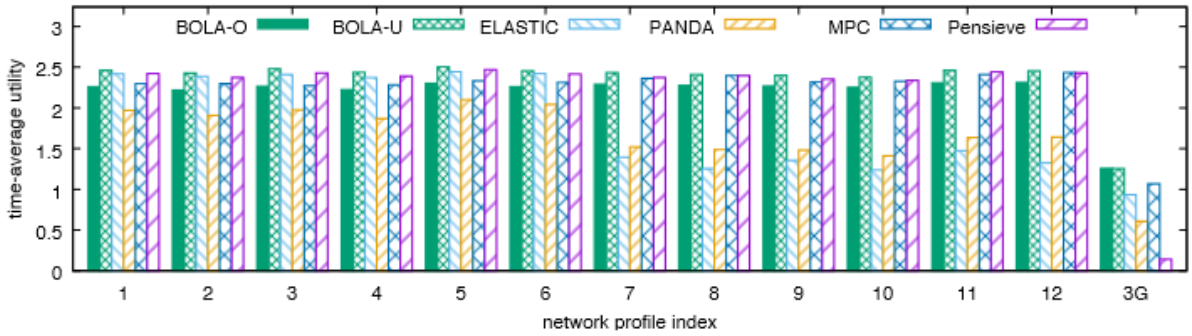
Gambar 4.5 menunjukkan utilitas waktu rata-rata BOLA-FINITE selama 10, 30 dan 120 menit waktu pemutaran dengan $\gamma p = 5$. Dibandingkan dengan BOLA-BASIC pada Gambar 2.5, kita melihat bahwa utilitas waktu rata-rata jauh lebih dekat ke batas optimal offline. Manfaat dari penyesuaian ini juga terlihat saat buffer bertambah besar, karena tidak ada penurunan utilitas yang signifikan yang disebabkan oleh pengisian buffer dengan segmen bitrate rendah pada tahap awal video.

BAB 5
MENGHINDARI OSILASI BITRATE

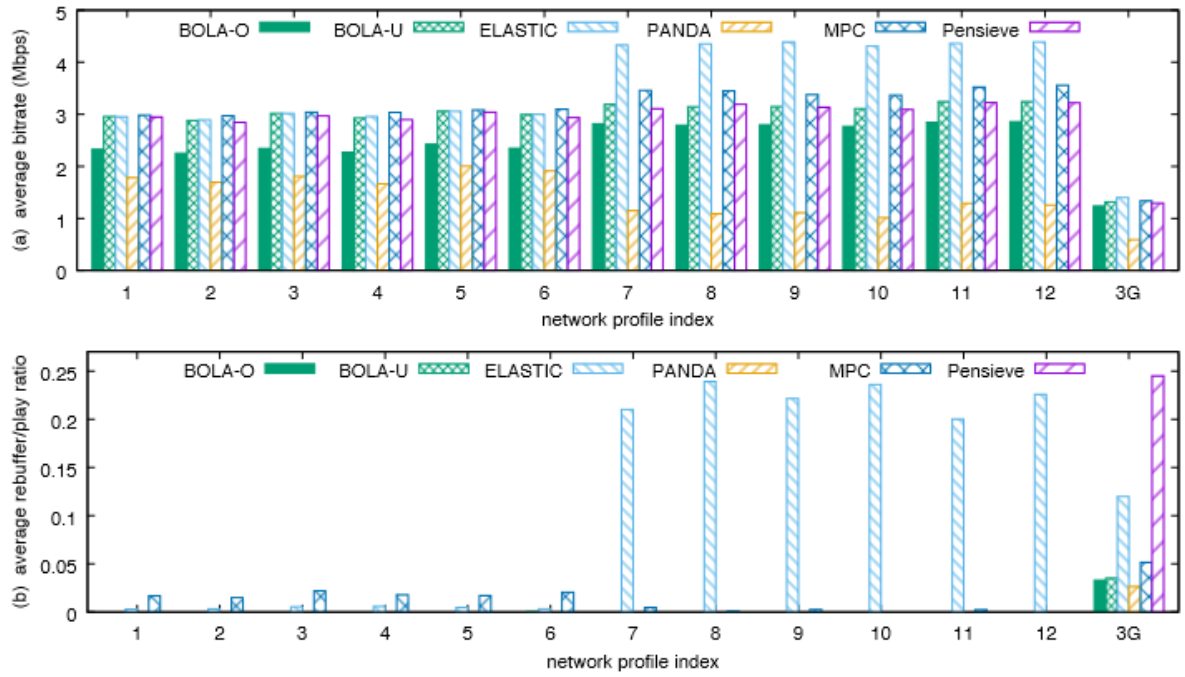
Sementara tujuan kinerja mengoptimalkan utilitas pemutaran dan kelancaran pemutaran, pengguna juga sensitif terhadap peralihan kecepatan bit yang berlebihan. Buku ini membahas tiga penyebab sakelar bitrate.

1) **Variasi bandwidth:** Saat kondisi jaringan berubah, pemutar memvariasikan bitrate, melacak bandwidth jaringan. Sakelar semacam itu dapat diterima; pemain tidak memiliki kendali atas bandwidth dan harus beradaptasi dengan kondisi jaringan yang berbeda.
2) **Ambang batas buffer padat:** Jumlah tingkat bitrate yang lebih besar dan/atau ukuran buffer yang lebih kecil dapat mendorong tingkat ambang lebih dekat. Jika perbedaan antara level ambang kurang dari durasi segmen p, menambahkan satu segmen yang diunduh ke buffer dapat mendorong level buffer ke beberapa level ambang sekaligus. Hal ini dapat menyebabkan BOLA-FINITE melakukan overshoot dan memilih bitrate yang terlalu tinggi untuk bandwidth yang tersedia. Akibatnya, pengunduhan segmen akan memakan waktu lebih dari p detik, menyebabkan penipisan buffer yang berlebihan, menyebabkan BOLA-FINITE menurunkan bitrate-nya lebih dari satu level. Dalam skenario seperti itu, BOLA-FINITE dapat berosilasi di antara bitrate, bahkan ketika bandwidth yang tersedia stabil.
3) **Kuantisasi bitrate:** Memiliki bandwidth jaringan yang stabil dan jarak ambang yang luas masih tidak menghindari semua peralihan bitrate. Misalkan bandwidth adalah 2,0 Mbps dan terletak di antara dua bitrate yang disandikan 1,5 dan 3,0 Mbps. Saat pemutar mengunduh segmen 1,5 Mbps, buffer terus bertambah. Saat buffer melewati ambang batas, pemutar beralih ke 3,0 Mbps, menghabiskan buffer. Setelah buffer cukup habis, pemutar beralih kembali ke 1,5 Mbps, dan siklus berulang. Dalam contoh ini, penonton mungkin lebih memilih pemutar video untuk tetap menggunakan bitrate 1,5 Mbps, mengorbankan beberapa utilitas agar memiliki osilasi yang lebih sedikit. Atau, pemirsa mungkin ingin memaksimalkan utilitas dan memutar sebagian video dalam bitrate lebih tinggi 3,0 Mbps dengan biaya osilasi lebih banyak. Penulis menjelaskan dua varian BOLA di bawah ini agar sesuai dengan kedua pemirsa.

Varian pertama yang penulis sebut BOLA-O meredakan osilasi dengan memperkenalkan pembatasan bitrate (baris 7-20 pada Gambar 5.1) saat beralih ke bitrate yang lebih tinggi. BOLA-O memverifikasi bahwa bitrate yang lebih tinggi berkelanjutan dengan membandingkannya dengan bandwidth yang diukur saat mengunduh segmen sebelumnya (baris 8–11). Karena motifnya adalah untuk membatasi osilasi daripada memprediksi bandwidth masa depan, adaptasi ini tidak menurunkan bitrate ke tingkat yang lebih rendah daripada unduhan sebelumnya (baris 12–13). Pengunduhan terus-menerus pada bitrate yang lebih rendah dari bandwidth akan menyebabkan buffer terus bertambah. BOLA-O menghindari ini dengan membiarkan buffer tergelincir ke ambang batas yang sesuai sebelum memulai pengunduhan (baris 15).



Gambar 5.1. Utilitas rata-rata waktu BOLA-O, BOLA-U, ELASTIC, PANDA, MPC dan Pensieve dengan γp = 5 memutar video 30 menit untuk profil jaringan uji DASH 1–12 dan jejak seluler (3G). Dibandingkan dengan ELASTIC dan PANDA, BOLA-U memiliki kegunaan sekitar 1,75 kali dari algoritme lain dalam kira-kira separuh kasus. MPC memiliki utilitas antara BOLA-O dan BOLA-U. Pensieve memiliki utilitas antara BOLA-O dan BOLA-U untuk profil 1–12 tetapi berkinerja lebih buruk untuk pelacakan seluler (3G).



Gambar 5.2. Membandingkan BOLA dengan ELASTIC, PANDA, MPC dan Pensieve menggunakan metrik mentah: bitrate rata-rata dan rasio rebuffer-to-play. BOLA, PANDA, dan Pensieve tidak melakukan buffer ulang untuk profil 1–12. ELASTIC hampir tidak memiliki rebuffering untuk profil 1–6, tetapi rasio rebuffer-to-play lebih besar dari 20% untuk profil 7–12. MPC memiliki beberapa rebuffering untuk hampir semua profil. Pensieve tidak memiliki buffer ulang untuk profil 1–12. Namun, Penseive memiliki rasio rebuffer-to-play 24% untuk pelacakan seluler (3G), karena tidak dapat bekerja dengan baik untuk kondisi bandwidth yang sangat berbeda dari set pelatihannya.

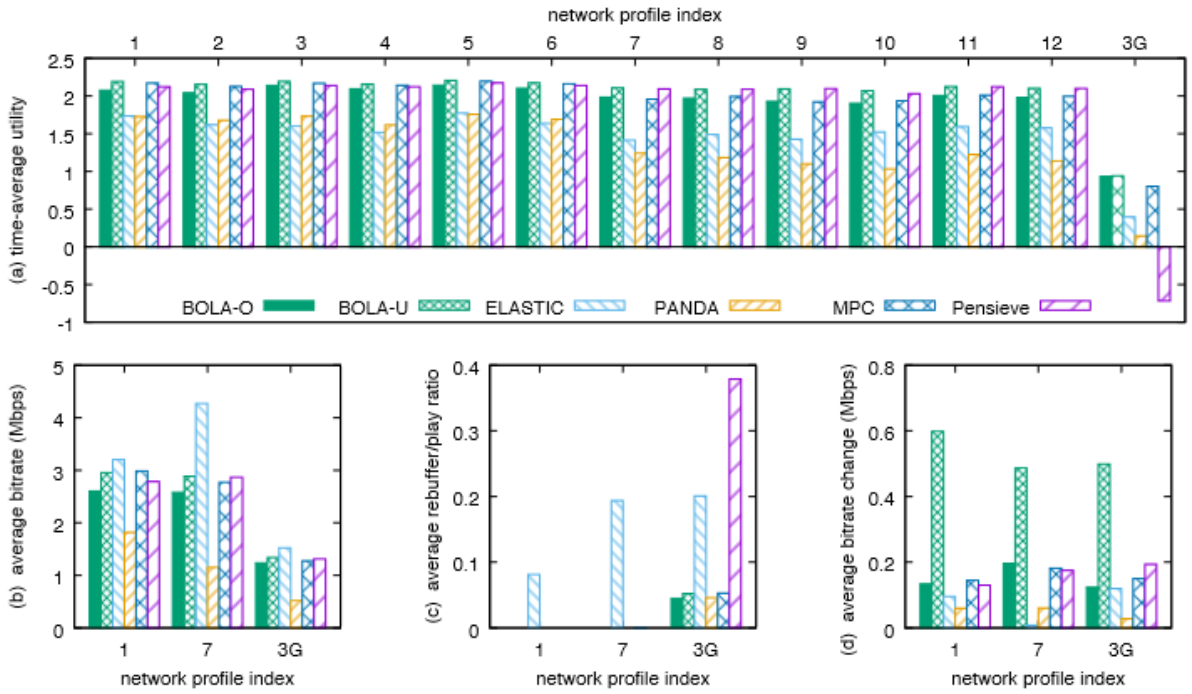
Varian kedua yang penulis sebut BOLA-U tidak mengorbankan utilitas. Pertumbuhan buffer yang berlebihan dihindari dengan membiarkan bitrate menjadi satu tingkat lebih tinggi dari bandwidth berkelanjutan (baris 17). Hal ini memungkinkan pemain untuk memilih 3 Mbps dalam contoh. Meskipun BOLA-U tidak menangani jenis osilasi ketiga, BOLA-U menangani jenis kedua yang lebih parah.

Melihat kembali Gambar 5.2, kita melihat bahwa stabilitas tambahan BOLA-U terbayar saat menggunakan ukuran buffer kecil dan BOLA-U mencapai utilitas yang lebih besar daripada BOLA-FINITE. Gambar 5.2 menunjukkan utilitas rata-rata waktu BOLA-O dan BOLA-U dengan $\gamma p = 5$ dan $Q_{max} p = 25s$ memutar video berdurasi 30 menit. Utilitas yang hilang dari BOLA-O untuk menghindari osilasi terlihat jelas. Dalam praktiknya, utilitas yang hilang dibatasi oleh jarak antara kecepatan bit yang disandikan; jika tingkat bitrate berikutnya yang lebih rendah tidak jauh dari bandwidth jaringan, maka utilitas kecil akan hilang.

Penulis mengukur osilasi dengan membandingkan segmen berurutan. Perubahan bitrate antara segmen dan berikutnya adalah perbedaan mutlak antara bitrate (dalam Mbps) dari dua segmen. Gambar 5.2 menunjukkan rata-rata perubahan bitrate di semua segmen. Sementara BOLA-U memiliki perubahan bitrate rata-rata yang tinggi karena kuantisasi, BOLA-O hanya mengganti bitrate karena variasi bandwidth jaringan.

5.1 PERBANDINGAN DENGAN ALGORITMA CANGGIH

Penulis sekarang membandingkan BOLA dengan empat algoritma canggih, ELASTIC, PANDA, MPC dan Pensieve. Pada buku ini menggunakan parameter desain default dan menguji BOLA-O dan BOLA-U. Meskipun BOLA berkinerja lebih baik dengan buffer yang lebih besar, penulis membatasi ukuran buffer hingga 25 detik untuk pengujian guna memastikan keadilan. ELASTIC menargetkan level buffer 15 detik tetapi level buffer bervariasi lebih tinggi. PANDA menargetkan tingkat buffer minimum 26 detik. Penulis menggunakan varian MPC RobustMPC dengan ukuran buffer 25 detik. MPC bergantung pada estimasi bandwidth; penulis menggunakan rata-rata harmonik selama lima segmen unduhan terakhir agar konsisten dengan metode evaluasi empiris. Buku ini melatih model jaringan saraf Pensieve untuk video dengan ukuran buffer 25 detik. Untuk melatih Pensieve, menggunakan pelacakan bandwidth yang dihasilkan menggunakan alat yang disediakan di repositori Pensieve seperti yang direkomendasikan.



Gambar 5.3. Utilitas rata-rata waktu BOLA-O, BOLA-U, ELASTIC, PANDA, MPC, dan Pensieve dengan γp = 5 memutar video berbeda selama 30 menit, menggunakan profil jaringan uji DASH 1–12 dan pelacakan seluler (3G). Perhatikan bahwa Pensieve memiliki utilitas 3G negatif karena rebuffering yang berlebihan (rata-rata rasio rebuffer-to-play adalah 38%). Metrik mentah juga disediakan dalam plot di atas.

Gambar 5.3 membandingkan algoritma menggunakan masing-masing dari 12 profil jaringan dan jejak seluler. BOLA-U secara konsisten berkinerja jauh lebih baik daripada PANDA. Sementara BOLA-U dan ELASTIC bekerja sama untuk profil 1–6, BOLA-U bekerja lebih baik secara signifikan untuk profil lain yang memiliki variasi bandwidth lebih besar. MPC dan Pensieve secara konsisten memperoleh utilitas antara BOLA-O dan BOLA-U untuk profil 1–12, tetapi berkinerja lebih buruk untuk pelacakan seluler. Penulis ulangi perbandingan menggunakan bitrate rata-rata dan metrik rebuffering pada Gambar 5.3. Ini memberikan wawasan tentang kekuatan dan kelemahan dari berbagai algoritma.

Membandingkan BOLA-U dengan ELASTIC: Untuk profil 1–6, BOLA-U memiliki bitrate yang kira-kira sama dengan ELASTIC. ELASTIC memiliki bitrate yang lebih tinggi untuk profil 7–12, tetapi hal itu menimbulkan biaya yang signifikan dalam hal buffering ulang. Untuk profil ini, rasio waktu rebuffering terhadap waktu putar lebih dari 20% untuk ELASTIC, sedangkan BOLA-U tidak memiliki rebuffering. Untuk pelacakan seluler, ELASTIC memiliki bitrate sedikit lebih tinggi daripada BOLA-U tetapi memiliki rasio buffer-to-play 12,0% dibandingkan dengan BOLA-U 3,5%. ELASTIC rebuffer lebih signifikan karena tidak bereaksi pada saat bandwidth turun.

Membandingkan BOLA-U dengan PANDA: Kedua algoritma tidak melakukan rebuffer untuk profil 1–12. Untuk jejak seluler, BOLA-U dan PANDA memiliki rasio rebuffer-to-play masing-masing sebesar 3,5% dan 2,6%. Namun, PANDA memiliki bitrate yang jauh lebih

rendah daripada BOLA-U. Alasannya adalah PANDA lebih konservatif dan dalam beberapa kasus tidak berubah ke bitrate yang lebih tinggi meskipun berkelanjutan.

Membandingkan BOLA-U dengan MPC: Kedua algoritme memiliki bitrate rata-rata yang serupa tetapi MPC memiliki bitrate sedikit lebih tinggi untuk beberapa profil. Namun, sementara BOLA-U tidak melakukan rebuffering, MPC memiliki beberapa rebuffering untuk sebagian besar profil. Meskipun dimungkinkan untuk menyetel parameter MPC untuk menghindari rebuffering tersebut, tidak jelas bagaimana memilih parameter yang bekerja secara konsisten untuk kondisi jaringan yang berbeda. Faktor lain yang mungkin berkontribusi pada MPC rebuffering adalah estimasi bandwidth. Ketika ada penurunan bandwidth yang besar, penaksir bandwidth rata-rata harmonik yang direkomendasikan memerlukan beberapa saat untuk bereaksi. Meskipun faktor RobustMPC dalam kesalahan estimasi jaringan, rebuffering tidak sepenuhnya dihilangkan.

Membandingkan BOLA-U dengan Pensieve: Untuk profil 1–12, Pensieve memperoleh utilitas antara BOLA-O dan BOLA-U, tetapi secara konsisten lebih dekat dengan BOLA-U. Namun, Pensieve memiliki terlalu banyak rebuffering di jejak seluler, menghasilkan utilitas yang jauh lebih buruk untuk jejak ini. Sementara pelacakan jaringan yang digunakan untuk melatih Pensieve menyertakan periode dengan bandwidth rendah yang mirip dengan pelacakan seluler, Pensieve tidak mempelajari model yang akan bekerja dengan baik dalam situasi bandwidth yang relatif rendah dalam pengaturan seluler. Ini menunjukkan kelemahan Pensieve karena tidak dapat beradaptasi dengan kondisi bandwidth yang berbeda secara signifikan dari set pelatihan.

Pada Gambar 5.3 menunjukkan hasil penulis untuk metrik sekunder osilasi bitrate penulis. BOLA-U tidak bekerja dengan baik dalam metrik ini, karena berusaha memaksimalkan utilitas dengan mengorbankan peningkatan osilasi. Membandingkan BOLA-O dengan ELASTIC, PANDA, MPC, dan Pensieve, ELASTIC memiliki perubahan rata-rata yang lebih rendah daripada BOLA-O hanya dalam kasus di mana ia memiliki reaksi yang lambat dan rebuffering yang berlebihan. PANDA memiliki perubahan rata-rata yang lebih rendah karena lebih konservatif dan dalam beberapa kasus tidak berubah ke bitrate yang lebih tinggi meskipun bitrate tersebut berkelanjutan. MPC memiliki perubahan rata-rata yang lebih tinggi daripada BOLA-O untuk profil 1–12. Pensieve memiliki perubahan rata-rata yang mirip dengan BOLA-O untuk profil 1–12.

Penulis juga menguji algoritme dengan lebih banyak video untuk menyelidiki kinerja saat mengubah karakteristik seperti jenis konten, durasi segmen, dan kecepatan bit yang tersedia. Tes menunjukkan hasil yang serupa. Salah satu contohnya adalah video yang disediakan dengan Pensieve. Video tersebut memiliki 49 segmen dengan durasi segmen 4 detik. Itu dikodekan pada enam bitrate: 0,3, 0,75, 1,2, 1,85, 2,85 dan 4,3 Mbps. Gambar 5.2 menunjukkan utilitas dan metrik untuk enam algoritma yang sama dengan kesimpulan yang mirip dengan Gambar 4.4 dan 4.5. Perhatikan bahwa Pensieve kembali gagal bekerja dengan baik pada pelacakan seluler, karena sangat berbeda dari set pelatihannya.

Dengan demikian, dari analisis empiris penulis, penulis dapat menyimpulkan bahwa BOLA mencapai utilitas yang lebih tinggi, dan bekerja lebih konsisten di berbagai skenario dibandingkan dengan ELASTIC, PANDA, MPC, dan Pensieve. Salah satu alasan konsistensi BOLA

adalah karena tidak memiliki banyak parameter. BOLA memiliki dua parameter desain γ dan V , yang memiliki signifikansi intuitif seperti yang dibahas dalam Bagian 2.3.2, dan opsi apakah akan menukar beberapa utilitas untuk mengurangi osilasi atau tidak. Algoritme lain memiliki sejumlah parameter berbeda dan menyetel parameter untuk skenario tertentu mungkin membuat sistem kurang cocok untuk skenario lainnya. Selain itu, kemampuan BOLA untuk meninggalkan segmen selama pengunduhan dan memulai pengunduhan dengan kecepatan bit yang lebih rendah memungkinkan BOLA mencapai rebuffering yang jauh lebih sedikit daripada algoritme lainnya.

5.2 PENERAPAN PEMUTAR REFERENSI DASH

Setelah mengembangkan landasan teoretis untuk BOLA dan mengujinya dengan simulasi, penulis menerapkan BOLA dalam pengaturan produksi. Secara khusus, penulis menerapkan BOLA di dash.js, pemutar referensi DASH standar sumber terbuka. Melalui dash.js, BOLA sekarang digunakan dalam produksi oleh beberapa penyedia video besar dan jaringan pengiriman seperti Akamai, BBC, CBS, dan Orange. Penyebaran dalam produksi menghadirkan sejumlah tantangan baru seperti beroperasi dengan kapasitas buffer yang lebih kecil, menangani kejadian dengan benar seperti pengguna mencari titik berbeda dalam video, dan mentolerir penundaan yang disebabkan oleh pemutar video yang tidak terkait dengan kondisi jaringan. Teknik yang penulis terapkan untuk menangani tantangan baru ini dijelaskan di Bab selanjutnya.

Parameter BOLA

Satu tantangan penerapan melibatkan pemilihan parameter BOLA γ dan V . Penulis memberikan intuisi untuk memilih parameter di Bagian ini, tetapi penyedia video lebih akrab dengan memilih target tingkat buffer. Untuk tujuan ini, sekarang kita membahas bagaimana menurunkan γ dan V dari kebutuhan intuitif. Pertimbangkan persyaratan berikut:

1. Penulis menginginkan Qmax level buffer maksimum.
2. Penulis ingin mengunduh dengan kecepatan bit tertinggi saat tingkat buffer adalah Q_{max}.
3. Penulis ingin mengunduh pada kecepatan bit terendah saat tingkat buffer kurang dari ambang Q_{Low}, dan penulis ingin mengunduh pada kecepatan bit yang lebih tinggi ketika tingkat buffer melampaui ambang batas.

Persyaratan ini mudah dipahami oleh penyedia video yang mungkin belum familiar dengan BOLA. Faktanya, penyedia video biasanya memiliki Q_{max} tingkat buffer maksimum yang disukai. Selanjutnya, mereka mungkin memiliki preferensi untuk Q rendah.

Untuk memenuhi persyaratan 1–2, kita ingin beralih dari memilih $a_M = 1$ menjadi memilih $\Sigma_{am} = 0$ pada ambang saat tingkat penyangga adalah Q_{max} Ini terjadi jika

$$\rho_{a_M=1} = \rho_{a=0}$$

$$\frac{V(v_M + \gamma p) - Q_{\max}}{S_M} = 0$$

Perhatikan bahwa BOLA memenuhi persyaratan 2 dan mengunduh dengan kecepatan bit tertinggi tepat sebelum ambang batas Q_{max} karena pada tingkat penyangga tersebut kita mendapatkan $pa_M = 1 <= pa_M = 1$ untuk $m < M$. Hal ini diilustrasikan pada Gambar 5.2.

Untuk memenuhi persyaratan 3, kita ingin beralih dari memilih $a_1 = 1$ menjadi memilih $a_2 = 1$ pada ambang saat tingkat buffer Q rendah. Ini terjadi jika

$$\rho_{a_1=1} = \rho_{a_2=1}$$

$$\frac{V(v_1 + \gamma p) - Q_{\text{low}}}{S_1} = \frac{V(v_2 + \gamma p) - Q_{\text{low}}}{S_2}$$

Dari persamaan tersebut kita peroleh

$$V = \frac{Q_{\max} - Q_{\text{low}}}{v_M - \alpha}, \quad \gamma p = \frac{v_M Q_{\text{low}} - \alpha Q_{\max}}{Q_{\max} - Q_{\text{low}}}$$

Dimana

$$\alpha = \frac{S_2 v_1 - S_1 v_2}{S_2 - S_1}.$$

Saat menghitung parameter BOLA dari Q_{low} dan Q_{max}, intuisi sebelumnya tentang γ dan V masih berlaku. Jika penyedia video memilih Q_{low} yang lebih besar, γ akan lebih besar dan BOLA akan memberi bobot lebih pada rebuffering. Jika penyedia video memilih Q_{max} yang lebih besar, V akan lebih besar.

5.3 KENDALA YANG DIHADAPI

Ada banyak pekerjaan baru-baru ini pada algoritma adaptasi bitrate, banyak yang didasarkan pada perkiraan bandwidth koneksi jaringan. FESTIVE menggunakan estimator bandwidth harmonis untuk memprediksi bandwidth masa depan dari unduhan sebelumnya, membatasi perubahan bitrate ke satu tingkat antara segmen yang berurutan untuk stabilitas.

Khususnya, FESTIVE berupaya menemukan pertukaran antara efisiensi dan keadilan dengan unduhan yang bersaing. BBA adalah algoritma berbasis buffer. BOLA memiliki beberapa kemiripan dengan BBA tetapi fungsi pemetaan dari level buffer ke bitrate video berbeda. Selain itu, BBA mengasumsikan bahwa ukuran buffer besar (dalam hitungan menit), sehingga membuatnya tidak cocok untuk video pendek. Selanjutnya, itu tidak memberikan jaminan teoretis apa pun untuk pendekatan berbasis penyangga. Algoritma yang terkenal adalah ELASTIC yang menggunakan teori kontrol untuk menyesuaikan bitrate sehingga menjaga hunian buffer pada tingkat yang konstan.

Algoritma penting lainnya adalah PANDA yang juga memperkirakan bandwidth jaringan. PANDA menjatuhkan bitrate unduhan segera setelah bandwidth rendah terdeteksi tetapi hanya meningkatkan bitrate secara perlahan untuk menyelidiki kapasitas sebenarnya ketika bandwidth yang lebih tinggi terdeteksi. Seperti FESTIVE, PANDA memperdagangkan efisiensi demi keadilan. Sebuah algoritma menggunakan model predictive control (MPC)

diusulkan untuk mengoptimalkan seperangkat metrik yang komprehensif. Dalam pendekatan ini, bitrate untuk segmen saat ini dipilih berdasarkan prediksi bandwidth jaringan untuk beberapa segmen berikutnya. Namun, kinerjanya bergantung pada keakuratan prediksi tersebut. Pendekatan ini juga memerlukan pengoptimalan offline yang signifikan untuk dilakukan di luar klien untuk rangkaian skenario yang lengkap.

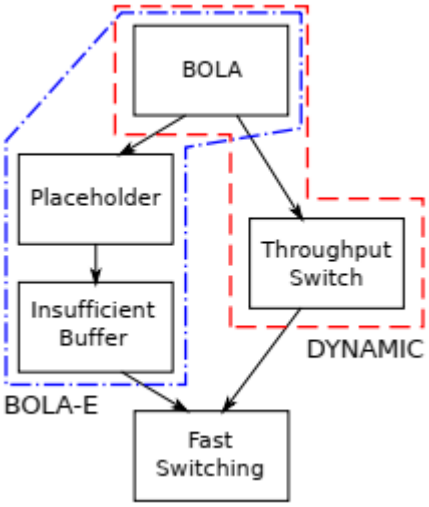
Model jaringan saraf dapat dilatih untuk video menggunakan ukuran buffer tertentu, menggunakan fungsi QoE sebagai hadiah. Satu set jejak bandwidth digunakan sebagai data pelatihan. Sayangnya, model yang dilatih tidak dapat ditransfer dengan mudah ke video lain atau, yang lebih penting, ke kondisi bandwidth yang tidak terwakili dalam data pelatihan. Tidak seperti pekerjaan sebelumnya, penulis memperoleh algoritme berbasis buffer dengan jaminan teoretis yang mudah diimplementasikan dalam klien dan penulis secara empiris menunjukkan kemanjurannya pada jejak jaringan yang luas. Metode yang disebut Oboe untuk menyetel otomatis parameter BOLA dan MPC disajikan dan terbukti meningkatkan kedua algoritme. Selanjutnya, pekerjaan menunjukkan bahwa Oboe digunakan bersama dengan algoritma ABR tradisional berkinerja lebih baik daripada ABR berbasis pembelajaran penguatan seperti Pensieve.

Buku ini memformulasikan adaptasi bitrate video untuk streaming ABR sebagai masalah maksimalisasi utilitas dan menurunkan BOLA, algoritme kontrol online yang terbukti mendekati optimal. Selanjutnya, penulis secara empiris menunjukkan kemanjuran BOLA menggunakan jejak ekstensif. Secara khusus, penulis menunjukkan bahwa algoritme online penulis mencapai utilitas yang mendekati algoritme offline optimal. Dalam buku ini menunjukkan bahwa algoritme penulis bekerja lebih baik daripada algoritme canggih dalam sejumlah skenario pengujian yang berbeda.

BAB 6
MENINGKATKAN ADAPTASI BITRAT PADA PEMAIN REFERENSI DASH

BOLA adalah satu-satunya algoritme ABR online yang diketahui dengan jaminan optimalitas yang dapat dibuktikan dalam kerangka kerja utilitas. Sementara BOLA mencapai utilitas yang hampir optimal dalam kondisi mapan, teori tersebut tidak berlaku untuk kondisi transien. Model teoretis umumnya tidak dapat menangkap semua kompleksitas sistem produksi, tetapi pendekatan penulis adalah memulai dengan landasan teoretis yang kuat yang disediakan oleh BOLA dan kemudian menyesuaikannya dengan implementasi praktis yang memodelkan seluk-beluk pengaturan produks.

Meskipun BOLA memberikan bitrate tinggi tanpa buffering atau osilasi yang signifikan, BOLA tidak memenuhi beberapa persyaratan yang penting untuk implementasi produksi dunia nyata. Secara khusus, karena BOLA sebagian besar menggunakan level buffer untuk pengambilan keputusan, BOLA tidak merespons dengan cepat peristiwa pengguna seperti memulai dan mencari saat buffer mulai kosong. Itu juga tidak merespons dengan cukup cepat terhadap perubahan cepat dalam profil throughput jaringan. Selain itu, kinerjanya tidak cukup baik dalam konteks streaming langsung di mana persyaratan latensi rendah mengamanatkan buffer kecil. Kekurangan seperti itu tidak spesifik untuk BOLA dan biasa terjadi pada algoritma canggih lainnya yang dikenal seperti BBA. Untuk meningkatkan BOLA, penulis mengambil pendekatan berbeda yang ditunjukkan pada Gambar 6.1 dan dijelaskan di bawah ini.



Gambar 6.1. Gambaran umum implementasi desain dan produksi penulis dari algoritme ABR untuk dash.js.

Algoritma BOLA-E Penulis memperkenalkan gagasan segmen virtual yang tidak berisi data video. Penulis mengembangkan algoritme placeholder baru yang dengan bijaksana menambahkan dan menghapus segmen virtual untuk mengubah level buffer yang digunakan

oleh BOLA untuk keputusan pengalihan kecepatan bit. Algoritme placeholder secara signifikan meningkatkan daya tanggap BOLA terhadap peristiwa jaringan dan pengguna. Selanjutnya, penulis merancang aturan buffer tidak mencukupi yang membantu menghindari rebuffering saat level buffer rendah, terutama dalam situasi live streaming saat buffer kecil. BOLA dengan algoritme placeholder dan aturan buffer yang tidak mencukupi merupakan versi BOLA yang disempurnakan yang penulis sebut BOLA-E.

BOLA-E pertama kali dirilis sebagai versi eksperimental di dash.js versi 2.0.0 pada 12 Februari 2016. Versi stabil dirilis dalam versi 2.6.0 pada 1 September 2017 dan telah digunakan oleh penyedia video sejak saat itu. BOLA-E tidak diaktifkan di pemutar referensi DASH secara default, tetapi merupakan salah satu dari dua algoritme ABR opsional yang tersedia untuk penyedia video. Penulis menyajikan BOLA-E dan mengevaluasinya.

Algoritma DINAMIS Pendekatan lain untuk meningkatkan BOLA adalah dengan menggunakan algoritma ABR berbasis throughput saat level buffer rendah dan kemudian secara dinamis beralih ke BOLA saat level buffer tinggi. Alasan untuk pendekatan ini adalah bahwa ABR berbasis throughput berperforma lebih baik dalam situasi seperti pengaktifan dan pencarian saat buffer rendah atau kosong. Dan BOLA bekerja lebih baik saat level buffer cukup besar. DYNAMIC juga pertama kali dirilis sebagai bagian dari dash.js versi 2.6.0 pada 1 September 2017 dan telah digunakan oleh penyedia video sejak saat itu. DYNAMIC saat ini merupakan algoritme ABR utama di pemutar referensi DASH dan diaktifkan secara default untuk penyedia video. Penulis menyajikan DINAMIS dan mengevaluasinya.

ALGORITMA FAST SWITCHING Penulis mengembangkan teknik yang disebut FAST SWITCHING yang dapat digunakan dengan algoritma ABR apa pun untuk meningkatkan kualitas video dengan mengganti segmen bitrate rendah di buffer klien dengan segmen bitrate tinggi. Pertimbangkan situasi di mana klien nirkabel telah mengunduh urutan segmen video bitrate rendah saat konektivitas buruk. Misalkan sekarang konektivitas klien meningkat. FAST SWITCHING memungkinkan klien untuk mengganti segmen bitrate rendah dalam buffer dengan segmen bitrate lebih tinggi yang sekarang dapat diunduh dengan konektivitas yang ditingkatkan. Dengan demikian, FAST SWITCHING memungkinkan pengguna beralih ke tampilan berkualitas lebih tinggi lebih cepat dari yang seharusnya. Penulis menerapkan FAST SWITCHING di dash.js versi 2.2.0 pada 6 Juli 2016. FAST SWITCHING dapat diaktifkan oleh penyedia video bersama dengan algoritme ABR apa pun, termasuk DYNAMIC default atau BOLA-E opsional. Penulis menyajikan FAST SWITCHING dan mengevaluasinya.

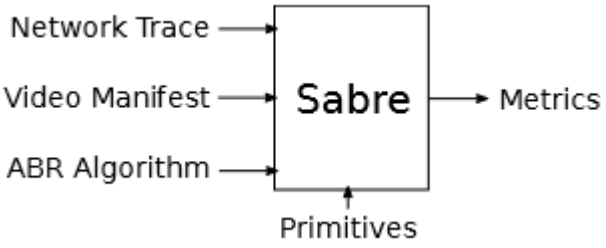
Penulis menerapkan dan mengevaluasi BOLA di pemain referensi DASH, dash.js. Pekerjaan penulis telah meningkatkan dash.js secara signifikan, seperti dicatat oleh komunitas dash.js. Dan, algoritme yang dijelaskan dalam bab ini digunakan secara aktif oleh penyedia video (termasuk Akamai, BBC, CBS, dan Orange) dalam produksi, karena mereka membuat pemutar video sendiri berdasarkan pemutar referensi standar.

6.1 SABRE: ALAT SUMBER TERBUKA UNTUK MENSIMULASIKAN LINGKUNGAN ABR

Alat simulasi yang akurat untuk ABR sangat penting untuk pengembangan algoritme. Namun hasil simulasi tidak berguna jika alat simulasi tidak mencerminkan kondisi praktis pemain. Penulis mengembangkan Sabre, alat yang akurat untuk mensimulasikan lingkungan

ABR yang dapat digunakan untuk merancang dan mengevaluasi algoritma ABR baru. Untuk akurasi simulasi, buku ini mendasarkan desain Saber pada arsitektur pemain referensi DASH, dash.js. Namun, pemutar video lain, seperti Pemutar Shaka Google dan pemutar HLS hls.js, secara fungsional mirip dengan dash.js, memungkinkan Sabre untuk digunakan sebagai alat yang efektif untuk mensimulasikan pemutar lain juga. Penulis menjadikan Sabre open source dan tersedia untuk umum bagi komunitas di GitHub, sehingga orang lain dapat menggunakan dan terus mengembangkan alat ini. Penulis juga menggunakan Sabre untuk mengevaluasi algoritme secara empiris yang disajikan dalam bab ini sebelum implementasi produksinya di dash.js.

Menggunakan Sabre menawarkan beberapa manfaat utama. Memutar video panjang dapat disimulasikan dalam waktu singkat, misalnya video satu jam dapat disimulasikan dalam waktu kurang dari satu detik. Selanjutnya, mudah untuk mensimulasikan kondisi jaringan yang sangat spesifik dengan cara yang andal dan dapat direproduksi. Selain itu, dimungkinkan untuk melakukan simulasi dalam skala besar menggunakan beberapa video dan ribuan jejak jaringan, seperti yang penulis lakukan dalam pekerjaan penulis.



Gambar 6.2. Sabre: Input, Output, dan Primitif.

Untuk mensimulasikan streaming video, Sabre mengaktifkan algoritme ABR sebelum mengunduh segmen. Algoritma ABR menyediakan bitrate dari segmen yang akan diunduh. Jika penggantian segmen diaktifkan, ini juga memberikan informasi apakah segmen yang akan diunduh adalah segmen baru atau pengganti segmen yang sudah ada. Saat segmen diunduh, Sabre mengumpulkan dan secara berkala melaporkan metrik ke algoritme ABR untuk digunakan dalam pengambilan keputusannya. Mirip dengan dash.js, Sabre memungkinkan pengabaian segmen unduhan yang sedang berlangsung. Selanjutnya, dash.js menggunakan peristiwa progres XMLHTTPRequest yang disediakan oleh browser. Sabre mensimulasikan peristiwa kemajuan untuk memungkinkan simulasi strategi pengabaian segmen. Input ke Sabre dijelaskan di bawah ini.

(1) **Jejak Jaringan.** Sabre memerlukan pelacakan jaringan untuk mensimulasikan sesi video. Pelacakan harus memiliki urutan catatan di mana setiap catatan berisi durasi waktu, dan throughput jaringan serta latensi untuk durasi tersebut. Jejak memungkinkan simulasi yang dapat direproduksi dari kondisi jaringan dunia nyata, memfasilitasi perbandingan antara algoritma yang berbeda atau antara pengaturan yang berbeda untuk menyetel algoritma tertentu. Jejak jaringan dapat diukur dari sistem aktual atau dapat berupa sintetik.

(2) **Deskripsi Video.** Saber juga membutuhkan deskripsi video yang serupa dengan manifes DASH. Deskripsi video mencakup panjang segmen (dalam detik), kecepatan bit yang disandikan, dan matriks ukuran segmen $C[i,j]. 1 \leq i \leq N, 1 \leq j \leq M$, dengan N adalah jumlah total segmen dalam video dan M adalah jumlah bitrate yang disandikan. Nilai $C[i,j]$ merepresentasikan ukuran (dalam bit) segmen ke-i dari video yang disandikan pada bitrate ke-j. Dengan mengizinkan matriks ukuran segmen ditentukan, penulis memungkinkan Sabre untuk mensimulasikan video dengan kecepatan bit variabel (VBR) secara akurat. Perhatikan bahwa deskripsi video dapat mewakili video yang sebenarnya atau dibuat secara sintetik.

(3) **Algoritma ABR.** Algoritme ABR dipanggil sebelum mengunduh segmen baru. Algoritme dalam bab ini seperti BOLA-E dan DYNAMIC tersedia dengan perangkat lunak Sabre. Namun, pengguna juga dapat mengembangkan algoritme ABR mereka sendiri sebagai modul Python dan mengujinya dengan Sabre.

Output Sabre terus-menerus mengumpulkan dan melaporkan daftar peristiwa dan metrik terperinci seperti kecepatan bit, waktu pengunduhan, dan ukuran setiap segmen yang diunduh, durasi setiap peristiwa penyangga ulang, setiap perubahan dalam kecepatan bit saat segmen diputar, dan semua pengabaian dan penggantian segmen.

Keluaran Sabre menyertakan tiga metrik penting yang penulis gunakan di sepanjang bab ini. Rasio buffering ulang adalah sebagian kecil dari waktu yang dihabiskan sesi video dalam status buffering ulang. Rasio rebuffer sama dengan total waktu rebuffer dibagi dengan jumlah total waktu rebuffer dan total waktu pemutaran. Bitrate rata-rata adalah rata-rata bitrate segmen yang dikodekan untuk semua segmen yang dirender. Osilasi kecepatan bit rata-rata adalah perbedaan rata-rata dalam kecepatan bit segmen yang dirender secara berurutan. Artinya, osilasi rata-rata sama

$$\text{Oscillations} = \frac{1}{N-1}\sum_{i=1}^{N-1} |\text{bitrate(i)} - \text{bitrate(i+1)}|$$

di mana bitrate(i) adalah bitrate yang disandikan dari segmen ke-i yang dirender dan N adalah jumlah segmen.

Perhatikan bahwa bitrate itu sendiri mungkin tidak berbanding lurus dengan QoE. Misalnya, peningkatan QoE yang diperoleh dengan memutakhirkan video 1 Mbps ke video 2 Mbps jauh lebih besar daripada peningkatan QoE yang diperoleh dengan memutakhirkan video 10 Mbps ke video 11 Mbps, meskipun peningkatan bitrate-nya sama. Namun, hubungan bitrate-ke-QoE umumnya monoton dan peningkatan yang satu akan meningkatkan yang lain. Jadi, penulis menggunakan bitrate sebagai ukuran QoE di seluruh bab ini.

Sabre juga menyediakan primitif yang menangkap fungsi umum yang dapat digunakan oleh pengembang algoritme ABR. Saat ini, penulis hanya menawarkan tiga primitif estimasi throughput. Primitif ini menghasilkan perkiraan throughput jaringan berdasarkan riwayat waktu pengunduhan segmen sebelumnya. Primitif throughput sliding-window menghasilkan estimasi dengan rata-rata throughput yang dicapai untuk k unduhan segmen yang berhasil di

masa lalu, di mana k adalah ukuran window yang ditentukan oleh pengguna. Primitif throughput jendela eksponensial menghasilkan perkiraan dengan merata-ratakan unduhan sebelumnya secara eksponensial dengan waktu paruh λ, di mana λ dapat ditentukan oleh pengguna. Penulis juga mendukung primitif throughput eksponensial ganda yang menggunakan primitif throughput jendela eksponensial dengan waktu paruh λ_1 dan λ_2 dan mengambil yang lebih kecil dari dua estimasi. Penulis mendukung primitif ini karena digunakan di Google Shaka Player dan di HLS player open source hls.js. Implementasi Sabre cukup modular bagi pengguna untuk menyediakan throughput tambahan atau primitif lainnya.

Sabre tidak mensimulasikan protokol level rendah seperti TCP, dan bergantung pada jejak unduhan yang dikumpulkan oleh pemain selama pengujian di dunia nyata. Selain itu, Sabre tidak mensimulasikan detail implementasi tingkat rendah seperti perilaku sebenarnya dari buffer Ekstensi Sumber Media browser. Namun, menghilangkan tingkat detail tersebut tidak memengaruhi performa algoritme ABR secara signifikan.

Sabre tidak mensimulasikan audio. Standar DASH mengharuskan video dan audio dikirimkan secara terpisah, dan pengunduhan audio biasanya terjadi pada sesi TCP yang berjalan paralel dengan pengunduhan video. Sekali lagi, mensimulasikan interaksi secara akurat memerlukan simulasi protokol tingkat rendah. Di sisi lain, ukuran aliran audio biasanya hanya sebagian kecil dari ukuran aliran video, memungkinkan solusi sederhana. Perhatikan contoh video yang disertai dengan audio 160 kbps. Penulis dapat mengurangi bandwidth jaringan yang tersedia sebesar 160 kbps di seluruh jejak jaringan untuk mensimulasikan video di Sabre.

Jejak jaringan yang digunakan dengan Sabre dalam pekerjaan penulis

1. Jejak 3G. Pada buku ini menggunakan jejak 3G dari [43], kumpulan 86 jejak yang dikumpulkan di Norwegia menggunakan koneksi 3G/HSDPA dalam perjalanan dengan bus, metro, trem, feri, mobil, dan kereta api. Jejak memiliki perincian 1 detik.
2. Jejak 4G. Pada buku ini menggunakan jejak 4G dari [56], kumpulan serupa dari 40 jejak yang dikumpulkan di Belgia menggunakan koneksi 4G/LTE pada perjalanan dengan sepeda, bus, mobil, kereta api, trem, dan berjalan kaki, dengan perincian 1 detik.
3. Jejak FCC. FCC menyediakan satu set jejak broadband publik. Pada buku ini memperoleh jejak throughput dari pengukuran dalam kategori penjelajahan web1, dengan setiap titik data mewakili throughput selama 5 detik. Jejak memiliki perincian 5 detik.

Tabel 6.1. Segmen Bitrate untuk Film Big Buck Bunny

SD	Mean	6.00	5.03	2.96	2.06	1.43	0.99	0.69	0.48	0.33	0.23
Bitrate (Mbps)	Std. Dev	1.08	0.89	0.56	0.39	0.28	0.18	0.12	0.10	0.05	0.04
HD	Mean	35.0	16.0	8.00	5.00	2.50	1.00				
Bitrate (Mbps)	Std. Dev	6.30	2.8	1.50	1.00	0.50	0.20				

Deskripsi video untuk Sabre dalam pekerjaan penulis

Penulis menggunakan Big Buck Bunny Movie [20], film berdurasi sepuluh menit, untuk simulasi penulis. Tabel 6.1 menunjukkan bitrate untuk deskripsi video SD dan HD, dengan

standar deviasi yang disebabkan oleh VBR. Penulis menggunakan pengkodean definisi standar (SD) yang merupakan pengkodean yang sama yang digunakan di Bab sebelumnya dengan sepuluh kecepatan bit mulai dari 230 kbps hingga 6 Mbps, dengan panjang segmen 3 detik. Masukan ke Sabre berisi ukuran dalam bit untuk setiap segmen $C[i, j]$. Penulis juga membuat deskripsi video definisi tinggi (HD) dengan enam bitrate2 mulai dari 1 Mbps hingga 35 Mbps dengan menskalakan ukuran segmen video SD yang diambil dari enam bitrate SD tertinggi. Dengan menggunakan penskalaan ini, penulis memperoleh bitrate HD dengan tetap mempertahankan variabilitas VBR.

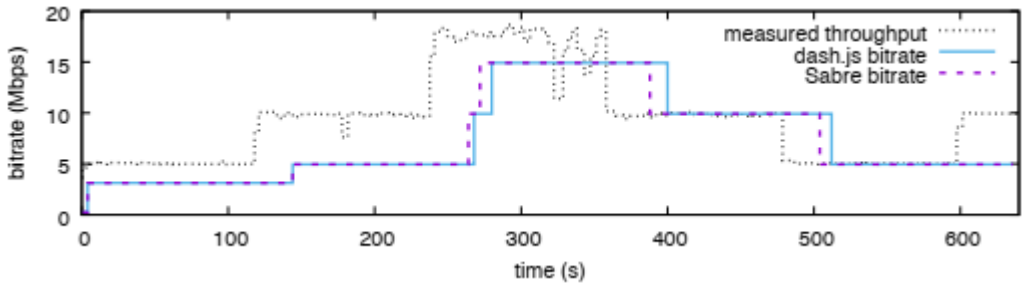
Validasi Sabre

Agar Sabre berguna selama pengembangan algoritme ABR, penulis perlu memastikan bahwa hasilnya secara akurat memprediksi hasil yang akan diperoleh oleh pemutar video dunia nyata yang sebenarnya. Di bagian ini penulis mengevaluasi seberapa akurat Sabre mengemulasi pemutar video dunia nyata seperti dash.js.

Penulis pertama kali menjalankan 20 sesi video di dash.js. Penulis menggunakan fungsi langkah untuk memodulasi throughput jaringan, menghabiskan dua menit masing-masing pada 5, 10, 20, dan 10 Mbps, lalu mengulang dari awal. Dengan memodulasi throughput secara berkala, penulis dapat menguji keakuratan Sabre dalam keadaan di mana algoritme ABR sering mengganti bitrate. Penulis menggunakan Big Buck Bunny Movie yang dapat dimuat di referensi dash.js player, video berdurasi 10 menit, 34 detik dengan pengkodean yang menggunakan sepuluh kecepatan bit mulai dari 250 kbps hingga 15 Mbps. Penulis memilih algoritma BOLA-E dan mengatur kapasitas buffer menjadi 25 detik. Untuk setiap sesi video, penulis merekam throughput seperti yang terlihat oleh pemutar dan tiga metrik QoE: rasio penyangga ulang, kecepatan bit rata-rata, dan osilasi kecepatan bit rata-rata.

Untuk membandingkan hasil penulis dari pemutar video dash.js dengan Sabre, penulis mensimulasikan 20 sesi video yang sama di Sabre menggunakan BOLA-E dan buffer 25 detik. Untuk memberi Sabre input deskripsi video yang cocok, penulis mengukur ukuran setiap segmen video. Kemudian, penulis membuat input pelacakan jaringan dari pengukuran throughput yang dibuat selama sesi dash.js yang sesuai. Setelah mensimulasikan setiap sesi, penulis membandingkan metrik QoE yang diberikan oleh simulasi Sabre dengan metrik QoE yang direkam dalam sesi dash.js yang sesuai.

Gambar 6.3 menunjukkan bitrate yang dipilih oleh algoritme ABR untuk sesi tipikal yang diukur pada dash.js dan Sabre. Pemutar yang sebenarnya (dash.js) dan pemutar yang disimulasikan (Sabre) menunjukkan perilaku pengalihan kecepatan bit yang serupa, karena kondisi jaringan berubah sesuai dengan throughput jaringan yang dimodulasi langkah. Tabel 6.2 menunjukkan kesalahan antara pengukuran QoE yang berasal dari dash.js dan sesi Sabre terkait.



Gambar 6.3. Kecepatan bit segmen yang diunduh dan diputar oleh pemutar dash.js dan oleh simulator Sabre untuk sesi biasa. Throughput yang ditampilkan adalah apa yang penulis ukur di dash.js untuk diputar ulang oleh Sabre.

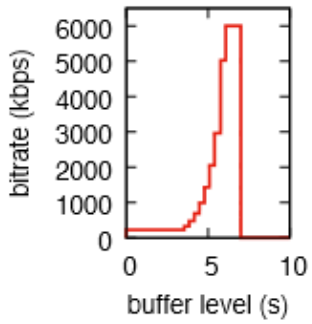
Tabel 6.2. Kesalahan dalam metrik QoE antara dash.js dan pengukuran Sabre terkait

Network Condition	Rebuffer Ratio Error		Average Bitrate % Error		Average Oscilation % Error	
	Mean	Std. Dev	Mean	Std. dev.	Mean	Std. Dev
Step	88×10^{-6}	307×10^{-6}	2.32	2.83	5.03	11.39
3G	14×10^{-3}	22×10^{-3}	1.88	1.68	5.25	7.010
4G	0	0	1.41	1.72	5.91	11.49

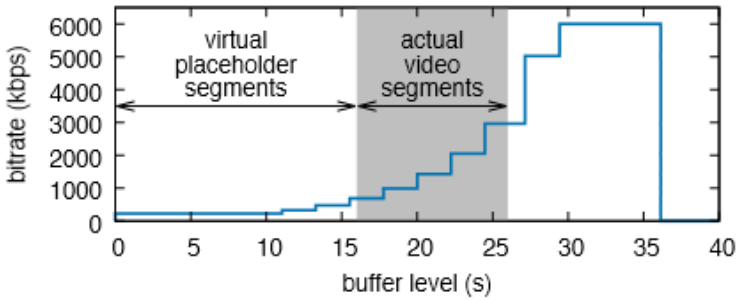
Bitrate rata-rata yang dilaporkan oleh Sabre memiliki error rata-rata 2,3%, sedangkan osilasi bitrate rata-rata memiliki error rata-rata 5,0%. Penulis juga menjalankan sesi pengujian dengan kondisi jaringan yang sesuai dengan pelacakan 3G dan 4G. Untuk jejak ini juga, metrik QoE yang diberikan oleh Saber sangat cocok dengan metrik yang diberikan oleh dash.js. Dengan demikian, Sabre menghasilkan metrik QoE yang secara akurat mencerminkan pengukuran dari pemain dash.js dunia nyata.

6.2 BOLA-E: PENINGKATAN BOLA

Algoritma ABR berbasis buffer seperti BOLA bekerja paling baik selama kondisi tunak, tetapi tidak terlalu responsif terhadap aktivitas pengguna seperti startup dan pencarian. Buffer biasanya kosong pada peristiwa ini, dan algoritme ABR berbasis buffer yang naif mungkin mengunduh banyak segmen dengan kecepatan bit lebih rendah sebelum mencapai tingkat buffer yang memadai untuk mengunduh dengan kecepatan bit berkelanjutan tertinggi. Sejumlah heuristik telah diusulkan untuk mengurangi startup lambat dalam algoritma ber basis buffer, tetapi heuristik masih jauh dari kinerja yang dicapai oleh algoritma berbasis throughput pada periode sementara. Pada Bagian ini penulis merancang dan mengimplementasikan algoritme placeholder sebagai penyempurnaan BOLA untuk mengatasi masalah ini.

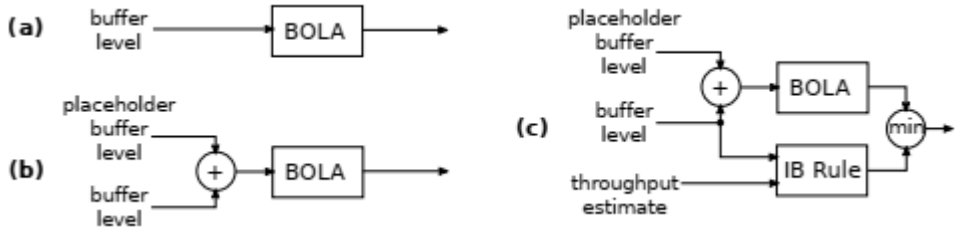


(a) Fungsi pemilihan bitrate BOLA untuk kapasitas buffer 10 detik. Perhatikan bahwa panjang segmen 3 detik berarti pengunduhan tidak diperbolehkan jika level buffer di atas 7 detik karena dapat menyebabkan buffer overflow.



(b) Fungsi keputusan bitrate BOLA-E untuk kapasitas buffer 10 detik setelah ekspansi buffer. Sementara buffer video masih terbatas pada 10 detik segmen aktual, segmen placeholder dapat menggeser jendela 10 detik, memungkinkan lebih banyak pemisahan antara ambang batas.

Gambar 6.4. Ekspansi penyangga memungkinkan BOLA-E memiliki pemisahan yang lebih besar antara ambang batas, mengurangi osilasi.



Gambar 6.5. Evolusi BOLA-E. (a) BOLA asli. (b) Menambahkan algoritme placeholder untuk BOLA-PL. (c) Menambahkan aturan buffer yang tidak mencukupi untuk BOLA-E.

Selanjutnya, algoritma berbasis buffer membutuhkan kapasitas buffer yang cukup untuk operasi yang stabil. Namun, hal ini tidak mungkin dilakukan untuk streaming langsung, seperti acara olahraga langsung, yang memerlukan latensi rendah. Dalam hal ini, kapasitas buffer harus lebih kecil dari batas latensi yang ingin kita capai. Jika kapasitas buffer kecil, ambang antara pilihan bitrate yang berbeda menjadi terlalu dekat. Pertimbangkan video biasa yang dikodekan pada sepuluh bitrate dengan panjang segmen 3 detik yang dialirkan ke pemutar video dengan kapasitas buffer 10 detik. Kapasitas penyangga ini memungkinkan pemisahan kurang dari 1 detik antara banyak ambang berurutan seperti yang terlihat pada Gambar 6.4(a). Bahkan variabilitas ukuran segmen yang kecil karena VBR dapat menyebabkan variabilitas pada level buffer. Dengan pemisahan kecil antara ambang batas, variabilitas tingkat buffer ini kemudian akan cukup untuk membuat algoritme ABR sering beralih antar bitrate, menyebabkan osilasi berlebihan. Pada sub bab setelah ini buku ini akan mengajarkan merancang dan menerapkan aturan buffer yang tidak mencukupi dan menggunakannya, bersama dengan ekspansi buffer, untuk mengatasi masalah ini. Gambar 6.5 secara grafis menunjukkan bagaimana penulis menyusun algoritma baru BOLA-E dari BOLA asli menggunakan algoritma placeholder, dan aturan buffer yang tidak mencukupi.

Algoritma Placeholder

Masalah mendasar dengan algoritme berbasis buffer adalah bahwa level buffer bukanlah proksi yang baik untuk throughput jaringan yang tersedia dalam situasi tertentu. Secara khusus, level buffer meremehkan atau tidak memberikan informasi tentang throughput saat ini ketika pengguna memulai atau mencari video. Faktanya, dalam kasus startup atau pencarian, buffer dimulai dengan kosong. Gagasan utama dari algoritme placeholder adalah bahwa level buffer dapat dibuat agar terlihat lebih besar dengan menyisipkan dan menghapus segmen placeholder virtual secara bijaksana dalam buffer, jika diperlukan. Tingkat buffer yang digunakan untuk keputusan ABR mencakup segmen placeholder dan video sebenarnya. Perhatikan bahwa segmen placeholder tidak memiliki konten video dan tidak dapat diputar. Mereka digunakan murni untuk memanipulasi tingkat buffer yang digunakan untuk pengambilan keputusan oleh algoritma ABR.

Algoritme placeholder meningkatkan respons terhadap memulai dan mencari acara dengan memasukkan segmen placeholder menggunakan langkah-langkah berikut.

1. Dapatkan estimasi throughput.
2. Pilih bitrate yang sesuai dengan estimasi throughput yang diperoleh di langkah (1).
3. Hitung tingkat buffer yang memungkinkan BOLA memilih bitrate yang dipilih. Untuk melakukannya, gunakan fungsi pemilihan bitrate yang digunakan oleh BOLA, seperti yang ditunjukkan pada Gambar 6.4(a). Kita dapat memilih tingkat buffer (sumbu x) yang sesuai dengan bitrate (sumbu y) yang dipilih pada langkah (2).
4. Masukkan cukup banyak segmen placeholder virtual ke dalam buffer untuk mendapatkan tingkat buffer yang diinginkan. Artinya, jumlah segmen placeholder yang disisipkan sama dengan tingkat buffer yang diinginkan dari langkah (3) dikurangi ukuran total segmen aktual dalam buffer.

Perhatikan bahwa algoritme perlu mendownload satu segmen bitrate rendah saat startup untuk mendapatkan estimasi throughput pada langkah (1) di atas. Namun, dalam kasus pencarian, itu sudah memiliki perkiraan bagus yang tersedia dari unduhan segmen sebelumnya.

Algoritme placeholder juga menghapus segmen placeholder saat situasi menuntut bitrate harus dipertahankan stabil dan tidak dinaikkan. Salah satu situasi tersebut adalah ketika BOLA melarang beralih ke bitrate ketika peralihan tersebut kemungkinan besar akan diikuti oleh peralihan ke bitrate yang lebih rendah dalam waktu singkat. Dalam situasi ini, algoritme placeholder mencoba mengurangi level buffer ke nilai yang sesuai dengan menghapus segmen placeholder.

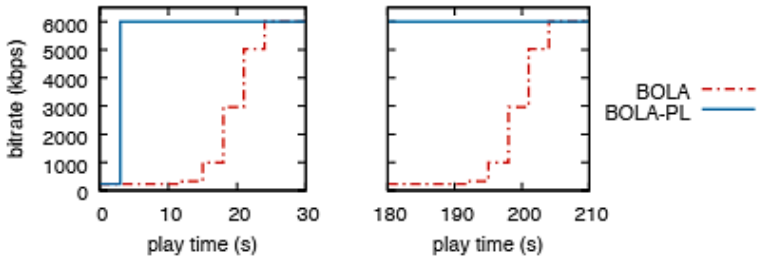
Evaluasi

Penulis sekarang mengevaluasi algoritme placeholder untuk respons terhadap peristiwa pengguna seperti startup dan pencarian. Pertama, penulis menggunakan jejak jaringan sintetik yang menjaga throughput relatif stabil pada 8 Mbps. Penulis menggunakan video SD yang dijelaskan. Penulis kemudian menggunakan Sabre untuk mengevaluasi BOLA tanpa algoritme placeholder dan BOLA-PL yang merupakan BOLA dengan algoritme placeholder. Kedua algoritma menggunakan kapasitas buffer 25 detik dalam evaluasi.

Idealnya, video harus mulai diputar secepat mungkin setelah peristiwa mulai/pencarian pada kecepatan bit 6 Mbps, yang merupakan kecepatan bit terenkode tertinggi dari video SD.

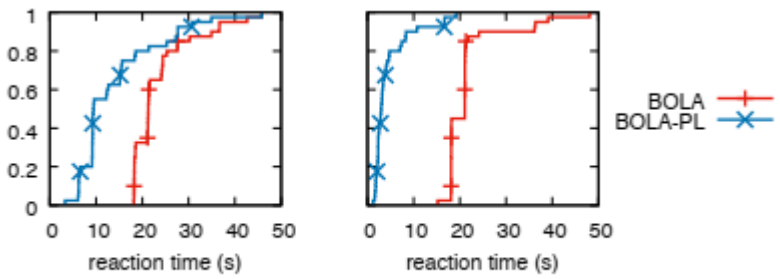
Gambar 6.6 mengevaluasi BOLA-PL dan BOLA untuk kejadian startup saat pengguna mengklik tombol mulai dan untuk kejadian pencarian di mana pengguna mencari titik 3 menit dalam video. BOLA-PL mulai bermain pada bitrate tertinggi pada 3,1 detik untuk skenario startup. Secara khusus, itu beralih ke kualitas tinggi dari segmen kedua dan seterusnya. Itu karena algoritma placeholder membutuhkan unduhan segmen pertama untuk mendapatkan estimasi throughput awal. Namun, BOLA-PL mulai bermain pada bitrate tertinggi mulai dari segmen pertama setelah pencarian, yaitu, pemutaran bitrate tinggi dimulai setelah 2,4 detik yang diperlukan untuk menyelesaikan pengunduhan segmen pertama. Di sisi lain, BOLA jauh kurang responsif untuk skenario startup dan seek karena harus menunggu level buffer naik sebelum beralih ke bitrate tertinggi. Secara khusus, dibutuhkan BOLA 24.1 untuk mengganti bitrate tertinggi untuk skenario startup dan seek.

Gambar 6.7 membandingkan startup dan mencari kinerja BOLA dan BOLA-PL untuk jejak 4G. Penulis mengulangi startup dan mencari eksperimen untuk video HD untuk masing-masing dari 40 jejak dan menghitung CDF dari waktu respons, di mana waktu respons adalah waktu yang diperlukan video untuk diputar pada bitrate berkelanjutan tertinggi. Waktu respons startup rata-rata untuk BOLA-PL adalah 9,3 detik, sedangkan BOLA membutuhkan waktu lebih lama untuk merespons pada 21,3 detik. Waktu respons pencarian rata-rata untuk BOLA-PL adalah 3,1 detik, lagi-lagi BOLA membutuhkan waktu lebih lama untuk merespons pada 21,1 detik.



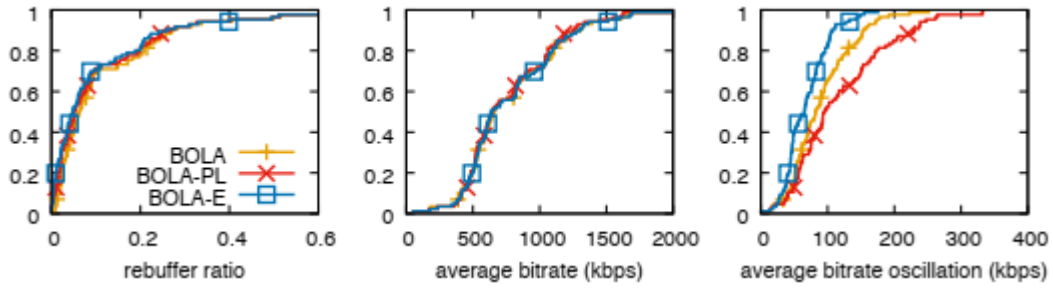
(a) Startup bitrate. (b) Seek bitrate.

Gambar 6.6. Bitrate pemutaran video sebagai fungsi dari waktu pemutaran video. BOLA dengan algoritme placeholder (BOLA-PL) bereaksi lebih cepat dengan mencapai bitrate berkelanjutan tertinggi dalam periode waktu yang jauh lebih singkat setelah startup atau pencarian daripada BOLA saja.



(a) Startup reaction time. (b) Seek reaction time.

Gambar 6.7. CDF waktu reaksi untuk BOLA versus BOLA-PL selama pengaktifan dan mencari 40 jejak jaringan 4G. BOLA-PL bereaksi jauh lebih cepat dan streaming pada bitrate berkelanjutan tertinggi lebih cepat daripada BOLA.



Gambar 6.8. CDF metrik QoE untuk BOLA, BOLA-PL, dan BOLA-E saat streaming video SD dengan kapasitas buffer 10 detik untuk 86 jejak 3G. BOLA-E secara signifikan mengurangi osilasi.

Aturan Penyangga Tidak Memadai

Buku ini mengusulkan solusi untuk masalah menghindari osilasi dalam streaming langsung latensi rendah. Persyaratan latensi rendah menyiratkan bahwa kapasitas buffer harus kecil. Untuk algoritme berbasis buffer apa pun seperti BOLA, ini berarti bahwa ambang di mana perubahan kecepatan bit dibuat saling berdekatan, dan bahkan varian kecil dalam ukuran segmen atau throughput jaringan dapat menyebabkan osilasi.

Menggunakan segmen placeholder memungkinkan pendekatan baru untuk masalah ini dengan membiarkan kapasitas buffer menjadi besar, namun tetap membatasi ukuran total segmen aktual dalam buffer agar tidak lebih dari nilai kecil. Artinya, penulis mengizinkan buffer besar dengan pemisahan yang signifikan antara ambang batas untuk peralihan bitrate. Namun, penulis membiarkan hanya sejumlah kecil segmen aktual yang disimpan di buffer, sisanya menjadi segmen placeholder. Dengan pendekatan ini, latensi tetap kecil, karena hanya segmen aktual yang berkontribusi pada latensi. Perhatikan bahwa karena hanya beberapa segmen sebenarnya yang dapat disimpan dalam buffer dengan ukuran yang jauh lebih besar, akan ada kejadian ketika terdapat cukup ruang dalam buffer dan throughput jaringan cukup tinggi untuk mengunduh segmen. Namun, algoritme harus dijeda karena segmen baru belum tersedia karena berada di luar jendela latensi. Dalam hal ini, segmen placeholder ditempatkan dalam buffer untuk menunjukkan bahwa segmen sebenarnya dapat diunduh jika segmen tersebut tersedia.

Penulis sekarang mengilustrasikan ekspansi buffer yang dijelaskan di atas dengan sebuah contoh. Gambar 6.4(a) menunjukkan contoh ambang peralihan bitrate BOLA untuk streaming langsung latensi rendah dengan kapasitas buffer 10 detik. Gambar 6.4(b) menunjukkan bagaimana ia dapat "ditarik" ke buffer yang lebih besar dengan memodifikasi parameter BOLA V dan γ. Ambang batas pada Gambar 6.4(b) berjarak paling tidak 2 detik, mengurangi potensi osilasi. Namun, ukuran total segmen aktual yang dapat disimpan dalam buffer masih paling banyak dari kapasitas buffer asli 10 detik.

Sayangnya, ekspansi buffer menghasilkan buffer besar dengan banyak segmen placeholder tetapi dengan sedikit segmen sebenarnya. Ini dapat meningkatkan buffering, meskipun mengurangi osilasi. Segmen placeholder mendorong BOLA-PL untuk mengunduh pada bitrate yang lebih tinggi seperti yang ditunjukkan pada Gambar 6.4(b), tetapi hal ini dapat menyebabkan buffer ulang saat segmen video habis, karena segmen placeholder bersifat virtual dan tidak dapat diputar. Penulis mengusulkan aturan buffer yang tidak mencukupi untuk mengatasi masalah rebuffering ini. Aturan tersebut memverifikasi setiap pilihan ABR oleh BOLA-PL untuk memastikan pengunduhan tidak mungkin menyebabkan peristiwa penyanggaan ulang menggunakan langkah-langkah berikut.

1. Kalikan estimasi throughput saat ini dengan 50% untuk mendapatkan throughput yang aman.
2. Lipat gandakan throughput aman dengan tingkat buffer video (tidak termasuk segmen placeholder) untuk mendapatkan ukuran unduhan yang aman.
3. Batasi pilihan ABR ke segmen dengan ukuran tidak lebih besar dari ukuran unduhan yang aman, selalu izinkan bitrate terendah.

Menggabungkan algoritme berbasis buffer dengan algoritme placeholder dan buffer yang tidak mencukupi menghasilkan algoritme hibrid yang memiliki keunggulan algoritme berbasis buffer sambil menghindari kelemahannya yang biasa.

Evaluasi

Penulis sekarang membandingkan BOLA-E yang menyertakan ekspansi buffer dan aturan buffer yang tidak mencukupi dengan BOLA-PL yang tidak menyertakan keduanya. Pertama, penulis mencatat bahwa penulis secara empiris mengonfirmasi bahwa daya tanggap BOLA-E untuk memulai dan mencari acara identik dengan BOLA-PL yang ditunjukkan pada Gambar 6.6 dan 6.7. Selanjutnya, penulis mengevaluasi algoritme ini pada pelacakan 3G dan 4G yang dijelaskan dengan kapasitas buffer 10 detik untuk mengevaluasi potensi buffer ulang dan osilasinya. Gambar 6.8 menunjukkan bahwa BOLA-PL dan BOLA-E memiliki rebuffering yang hampir sama dan perilaku bitrate rata-rata untuk jejak 3G. Namun, BOLA-E yang menggunakan ekspansi buffer dan aturan buffer yang tidak mencukupi memiliki osilasi yang jauh lebih sedikit daripada BOLA-PL. Secara khusus, BOLA-E memiliki osilasi bitrate rata-rata 65 kbps versus 95 kbps untuk BOLA-PL. Penulis juga menyertakan metrik untuk BOLA pada Gambar 6.8 untuk menunjukkan bahwa, sementara BOLA-E meningkatkan waktu reaksi, itu tidak menurunkan metrik QoE kondisi mapan. Faktanya, ini mengurangi osilasi bitrate. Perhatikan bahwa BOLA-PL tanpa ekspansi buffer dan tanpa aturan buffer yang tidak mencukupi meningkatkan osilasi bitrate jika dibandingkan dengan BOLA. Hasil empiris untuk jejak 4G serupa dengan jejak 3G dan penulis tidak memasukkannya di sini karena keterbatasan ruang.

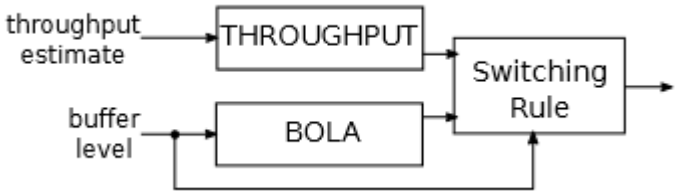
BAB 7
DINAMIS: BOLA DENGAN THROUGHPUT

Bab ini memperkenalkan peningkatan pada algoritma berbasis buffer BOLA untuk mengurangi masalah dengan startup, seek dan streaming latensi rendah dan membuat algoritma baru BOLA-E. Di bagian ini, penulis menjelaskan pendekatan yang berbeda untuk mengurangi masalah yang sama, yang mengarah ke pembuatan DINAMIS yang saat ini merupakan algoritme ABR default di pemutar referensi DASH, dash.js (lihat Gambar 7.1).

Penulis mengamati bahwa algoritme berbasis throughput bekerja dengan baik dalam situasi level buffer rendah, sedangkan algoritme berbasis buffer seperti BOLA berperforma lebih baik pada level buffer yang lebih besar. Dengan demikian, penulis mengusulkan algoritma DINAMIS yang menggunakan algoritma berbasis throughput sederhana yang disebut THROUGHPUT saat level buffer rendah (seperti saat startup dan seek event), dan menggunakan BOLA saat level buffer tinggi seperti yang ditunjukkan pada Gambar 7.1.

THROUGHPUT adalah heuristik sederhana yang pertama memperkirakan throughput jaringan dengan menggunakan sliding-window primitif dan kemudian mengambil bitrate terenkode tertinggi yang lebih rendah dari faktor keamanan 90% dari perkiraan throughput.

Algoritma DYNAMIC bekerja sebagai berikut. Saat memulai, DYNAMIC memulai dengan mengaktifkan THROUGHPUT. Pada tahap ini, BOLA masih lebih memilih bitrate yang terlalu rendah. Ketika level buffer mencapai 10 detik atau lebih dan BOLA memilih bitrate setidaknya setinggi bitrate yang dipilih oleh THROUGHPUT, DYNAMIC beralih ke BOLA. DYNAMIC beralih kembali ke THROUGHPUT ketika level buffer turun di bawah 10 detik dan BOLA memilih bitrate yang lebih rendah daripada THROUGHPUT.



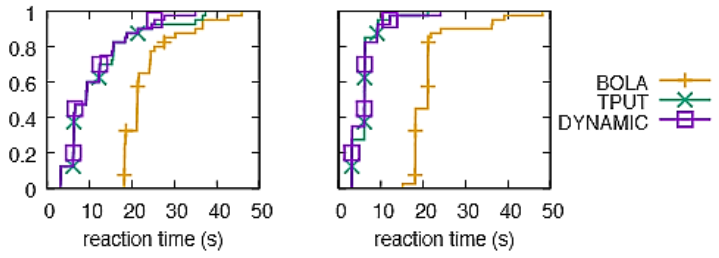
Gambar 7.1. Algoritma DINAMIS menggabungkan BOLA dan THROUGHPUT.

Pertama, penulis mempelajari waktu respons DYNAMIC sehubungan dengan BOLA dan THROUGHPUT untuk memulai dan mencari acara menggunakan buffer 25 detik. Gambar 7.2 memplot CDF waktu reaksi ketika algoritme ABR mencapai bitrate berkelanjutan tertinggi. Seperti yang diharapkan, baik THROUGHPUT dan DYNAMIC memberikan waktu respons yang cepat, sementara BOLA merespons lebih lambat karena perlu membangun buffer ke tingkat yang cukup untuk beralih ke bitrate berkelanjutan tertinggi. Perhatikan bahwa peningkatan waktu reaksi tidak menimbulkan degradasi dalam metrik QoE lainnya, seperti yang ditunjukkan pada Gambar 7.2. Gambar 7.2 membandingkan DYNAMIC, BOLA dan

THROUGHPUT secara individual untuk dua skenario pada 40 jejak 4G dan memplot CDF untuk rasio penyangga ulang, bitrate rata-rata, dan osilasi bitrate rata-rata. Skenario pertama, ditunjukkan pada Gambar 7.2(a), diturunkan dengan mensimulasikan video HD penulis melalui jejak 4G dengan kapasitas buffer 25 detik untuk meniru pengalaman menonton VOD pada umumnya. Dalam skenario ini, ketiga algoritme mencapai rasio penyangga ulang yang serupa. Namun, BOLA dan DYNAMIC mencapai throughput yang lebih besar daripada THROUGHPUT. Secara khusus, BOLA dan DYNAMIC masing-masing mencapai 19% dan 22% lebih banyak median throughput daripada THROUGHPUT. Selanjutnya, THROUGHPUT memiliki lebih banyak osilasi daripada BOLA atau DINAMIS. Secara khusus, pada persentil ke-90, BOLA dan DYNAMIC memiliki osilasi lebih sedikit masing-masing 1301 kbps dan 1421 kbps, sedangkan THROUGHPUT memiliki osilasi lebih tinggi pada 1929 kbps. Singkatnya, untuk pengaturan VOD biasa, BOLA dan DYNAMIC tampil lebih baik secara konsisten daripada THROUGHPUT.

Skenario kedua, ditunjukkan pada Gambar 7.2(b), mengevaluasi ketiga algoritme melalui jejak 4G dengan buffer kecil 10 detik untuk mensimulasikan skenario live streaming latensi rendah. Perhatikan bahwa dengan kapasitas buffer yang rendah ini, DYNAMIC tidak pernah dapat melewati ambang level buffer 10 detik untuk memilih BOLA. Dalam skenario ini, ketiga algoritme mencapai rasio penyangga ulang yang serupa. BOLA mencapai throughput yang lebih besar daripada THROUGHPUT dan DYNAMIC. Secara khusus, THROUGHPUT dan DYNAMIC mencapai 11% (yaitu, 2049 kbps) lebih sedikit throughput median daripada BOLA. Namun, throughput BOLA yang tinggi menyebabkan osilasi yang berlebihan. BOLA memiliki lebih banyak osilasi daripada THROUGHPUT dan DYNAMIC. Secara khusus, pada nilai median, THROUGHPUT dan DYNAMIC memiliki osilasi bitrate 1089 kbps, sedangkan BOLA memiliki osilasi lebih tinggi pada 2465 kbps. Singkatnya, untuk pengaturan live biasa, THROUGHPUT dan DYNAMIC tampil lebih baik secara konsisten daripada BOLA.

Kesimpulan utama yang dapat penulis tarik dari eksperimen penulis adalah bahwa, sementara BOLA bekerja lebih baik dalam skenario VOD dengan buffer yang lebih besar, dan THROUGHPUT bekerja lebih baik untuk skenario buffer yang lebih kecil seperti live streaming latensi rendah, DYNAMIC menggabungkan keunggulan keduanya dan bekerja dengan baik di keduanya. situasi. DYNAMIC juga memberikan respons yang cepat dalam skenario startup dan seek, menjadikannya pilihan yang baik secara keseluruhan sebagai algoritme ABR.



(a) Startup reaction time. (b) Seek reaction time.

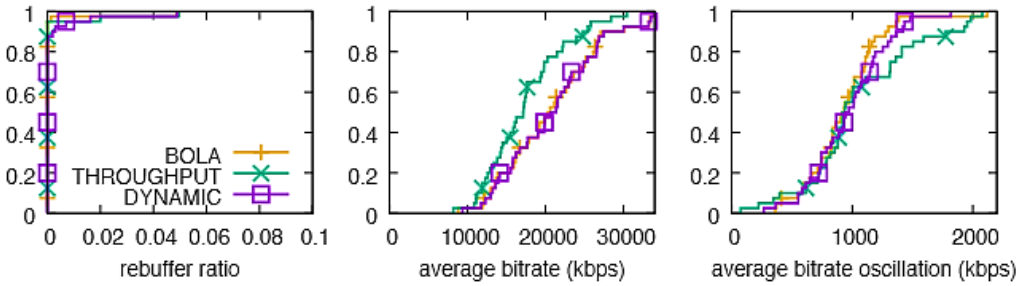
Gambar 7.2 . CDF waktu reaksi untuk BOLA versus THROUGHPUT versus DYNAMIC selama startup dan mencari 40 jejak jaringan 4G. THROUGHPUT dan DYNAMIC bereaksi

jauh lebih cepat dan streaming dengan bitrate berkelanjutan tertinggi lebih cepat daripada BOLA.

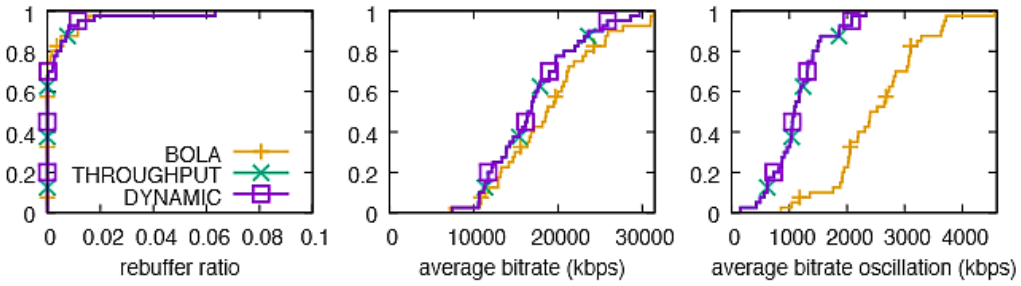
7.1 PERALIHAN CEPAT: Algoritma Penggantian Segmen

Buffer yang besar dapat meningkatkan stabilitas kinerja ABR karena dapat menyerap variasi kecil dalam kondisi jaringan, tetapi dapat menurunkan daya tanggap pemutar video terhadap peristiwa jaringan. Jika throughput jaringan tiba-tiba meningkat secara signifikan, algoritme ABR dapat mengunduh segmen pada kecepatan bit yang lebih tinggi. Namun, pemutar video harus terlebih dahulu memutar segmen kecepatan bit rendah yang sudah ada di buffer sebelum dapat merender segmen kecepatan bit tinggi yang baru diambil. Semakin besar kapasitas buffer, semakin banyak segmen bitrate rendah yang dapat ditampungnya, dan semakin lama menunggu sebelum pengguna dapat beralih ke kualitas yang lebih tinggi.

Penulis mengusulkan algoritme yang disebut FAST SWITCHING yang meningkatkan respons pemutar video ke throughput jaringan yang lebih tinggi dengan mengganti segmen yang sudah ada di buffer. Secara khusus, FAST SWITCHING memungkinkan penyedia video memiliki buffer yang lebih besar karena alasan stabilitas ABR, namun memiliki waktu respons yang lebih cepat terhadap peristiwa jaringan. Penulis menemukan bahwa FAST SWITCHING sangat berguna untuk penyedia video dengan konten VOD yang lebih panjang, seperti episode TV dan film, di mana diperlukan buffer yang lebih besar dan tidak ada persyaratan latensi rendah. Perhatikan bahwa FAST SWITCHING dapat digunakan dengan strategi pemilihan bitrate apa pun, termasuk BOLA-E, THROUGHPUT, dan DYNAMIC. Faktanya, implementasi dash.js saat ini memungkinkan FAST SWITCHING ditambahkan ke ketiga opsi.



(a) VOD streaming with a 25s buffer.



(b) Live streaming with a 10s buffer.

Gambar 7.3. DINAMIS menggabungkan kekuatan BOLA dan THROUGHPUT: Ini memiliki bitrate BOLA yang lebih tinggi untuk VOD dengan kapasitas buffer besar dan osilasi rendah THROUGHPUT untuk live streaming dengan kapasitas buffer kecil.

FAST SWITCHING berfungsi menggunakan langkah-langkah berikut.

1. Putuskan apakah akan mengunduh segmen baru atau segmen pengganti. Sebelum mengunduh segmen, algoritme memanggil algoritme pemilihan bitrate (mis., BOLA-E) untuk menentukan bitrate b yang dapat digunakan saat ini. Jika ada segmen dalam buffer dengan bitrate lebih rendah dari b dan jika segmen tersebut dapat diganti dengan aman, maka FAST SWITCHING memutuskan bahwa segmen berikutnya yang diunduh akan menjadi pengganti. Jika tidak, segmen berikutnya yang diunduh akan menjadi segmen baru yang ditambahkan ke akhir buffer. Secara intuitif, sebuah segmen tidak dapat diganti dengan aman jika terlalu dekat dengan play head dan kemungkinan besar akan mulai dimainkan sebelum penggantian dapat diunduh. Itu akan menghasilkan unduhan yang sia-sia. FAST SWITCHING menganggap setiap segmen yang dijadwalkan untuk mulai direnderdalam 1,5 × (panjang segmen) detik berikutnya tidak dapat diganti dengan aman.
2. Tentukan segmen mana yang akan diganti. Jika ditentukan pada langkah (1) bahwa segmen perlu diganti, FAST SWITCHING mengunduh pengganti untuk segmen paling awal di buffer yang dapat diganti dengan aman dan memiliki bitrate lebih rendah daripada bitrate saat ini b.

Pilihan 1,5 × (panjang segmen) dalam menentukan penggantian yang aman memberikan faktor keamanan 50% untuk memperhitungkan kemungkinan variasi waktu pengunduhan karena variabilitas jaringan dan/atau ukuran segmen. Perhatikan juga bahwa FAST SWITCHING menggantikan segmen dalam urutan paling awal-tenggat-pertama (EDF), dimulai dari segmen yang memiliki tenggat waktu paling awal untuk dimainkan (yaitu, paling dekat dengan play head). Pengurutan ini memungkinkan untuk mengganti lebih banyak segmen, karena segmen yang lebih rendah dalam urutan memiliki lebih banyak waktu untuk penggantian.

Algoritme FAST SWITCHING bekerja dengan algoritma ABR berbasis throughput dan berbasis buffer. Namun, algoritme berbasis buffer mungkin memerlukan penyesuaian. Saat FAST SWITCHING memilih untuk mengganti segmen yang ada, level buffer akan habis karena segmen sedang diputar, tetapi level buffer tidak akan ditambah oleh segmen yang diunduh. Level buffer yang lebih rendah ini dapat menyebabkan algoritme ABR berbasis buffer untuk memilih bitrate yang lebih rendah dan dengan demikian meningkatkan osilasi.

Penulis menangani masalah ini menggunakan dua pendekatan berbeda saat mengintegrasikan FAST SWITCHING dengan BOLA-E dan DYNAMIC. BOLA-E menyisipkan satu segmen placeholder di buffer setelah setiap penggantian segmen berhasil. Solusi ini tidak berfungsi untuk DYNAMIC karena tidak menggunakan algoritme placeholder. Alih-alih, DYNAMIC beralih ke THROUGHPUT setiap kali ada penggantian segmen, hingga level buffer stabil, setelah itu dapat beralih kembali ke BOLA.

Saat menggunakan FAST SWITCHING, pemutar membuang beberapa segmen dengan kecepatan bit lebih rendah dengan menggantinya dengan segmen dengan kecepatan bit lebih tinggi, meningkatkan total bit yang diunduh oleh klien. Dalam contoh SD di atas, ketika klien mengalami peningkatan 48% dalam rata-rata bitrate rata-rata, 10% bit diunduh dan dibuang oleh klien. Namun, bit diunduh dan dibuang hanya untuk waktu singkat ketika throughput

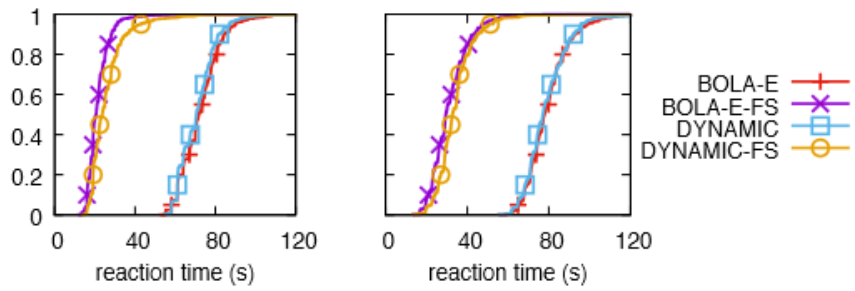
jaringan berubah secara drastis dan penggantian segmen diperlukan. Jadi, dampak FAST SWITCHING secara keseluruhan pada lalu lintas server-klien kurang signifikan.

Penulis sekarang mengevaluasi FAST SWITCHING dengan mengintegrasikannya dengan BOLA-E dan DYNAMIC. Untuk mensimulasikan FAST SWITCHING secara efektif, kita perlu membuat skenario di mana throughput jaringan meningkat

Untuk setiap pelacakan, beberapa saat setelah throughput jaringan meningkat, penonton mulai melihat video pada kecepatan bit yang lebih tinggi yang dapat dipertahankan oleh throughput yang lebih tinggi. Penulis mengukur waktu reaksi sebagai waktu yang berlalu sejak throughput jaringan meningkat hingga saat pengguna mulai melihat video pada kecepatan bit berkelanjutan tertinggi. Secara khusus, untuk video SD (resp., HD), penulis mengukur waktu hingga video mulai dirender pada 6 Mbps (resp., 16 Mbps). Dalam kedua kasus tersebut, penulis menyimulasikan pemutar video dengan buffer 25 detik.

Gambar 7.3 menunjukkan waktu reaksi untuk BOLA-E dan DYNAMIC dengan FAST SWITCHING, masing-masing dilambangkan dengan "BOLA-E-FS" dan "DYNAMIC-FS", dibandingkan dengan BOLA-E dan DYNAMIC sendiri. Kita dapat melihat bahwa PERALIHAN CEPAT meningkatkan waktu reaksi rata-rata sekitar 50 detik untuk algoritme ABR dan untuk video SD dan HD. Ini berarti bahwa pengguna akan melihat video dengan kualitas lebih tinggi sekitar 50 detik lebih awal dengan FAST SWITCHING dibandingkan tanpa FAST SWITCHING.

Perhatikan bahwa peningkatan 50 detik lebih dari kapasitas buffer 25 detik. Hal ini dimungkinkan karena ada dua komponen waktu reaksi untuk algoritma ABR tanpa FAST SWITCHING. Pertama, algoritme ABR perlu menentukan bahwa throughput telah meningkat dan memilih bitrate yang lebih tinggi. Kedua, segmen dengan kecepatan bit rendah yang sudah ada dalam buffer perlu diputar sebelum segmen dengan kecepatan bit lebih tinggi yang baru dapat diputar. PERALIHAN CEPAT mengurangi kedua komponen waktu reaksi, karena dapat menggantikan segmen bitrate rendah yang diunduh di kedua fase.

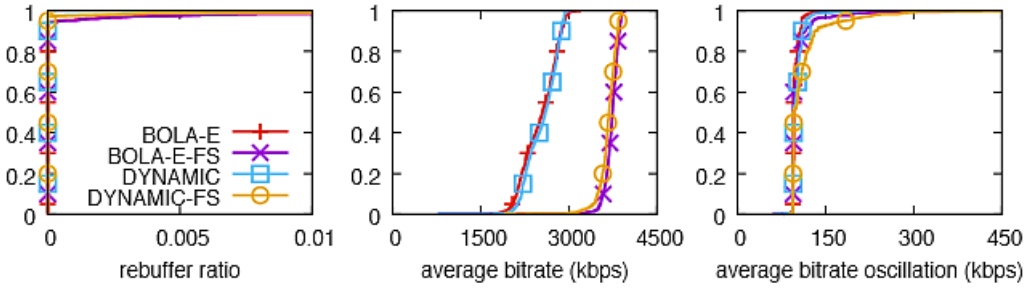


(a) SD video. (b) HD video.

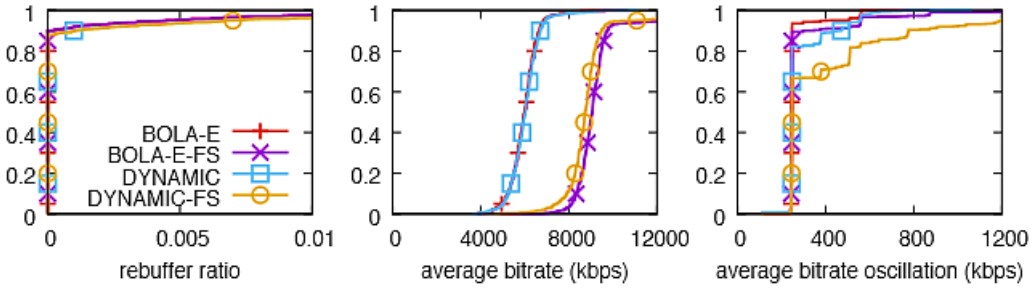
Gambar 7.4. CDF waktu reaksi saat meningkatkan throughput jaringan menggunakan BOLA-E dan DYNAMIC dengan dan tanpa FAST SWITCHING menggunakan buffer 25 detik. Waktu reaksi menunjukkan berapa lama waktu yang dibutuhkan untuk mulai merender pada bitrate berkelanjutan tertinggi setelah throughput jaringan meningkat.

Karena FAST SWITCHING beralih ke kecepatan bit yang lebih tinggi lebih cepat, penulis berharap FAST SWITCHING juga meningkatkan kecepatan bit rata-rata untuk eksperimen di atas. Bahkan, kolom tengah pada Gambar 7.4 menunjukkan peningkatan yang signifikan. Ini

meningkatkan bitrate rata-rata sekitar 45% untuk kedua algoritme ABR dan untuk video SD dan HD.



(a) Streaming the SD video.



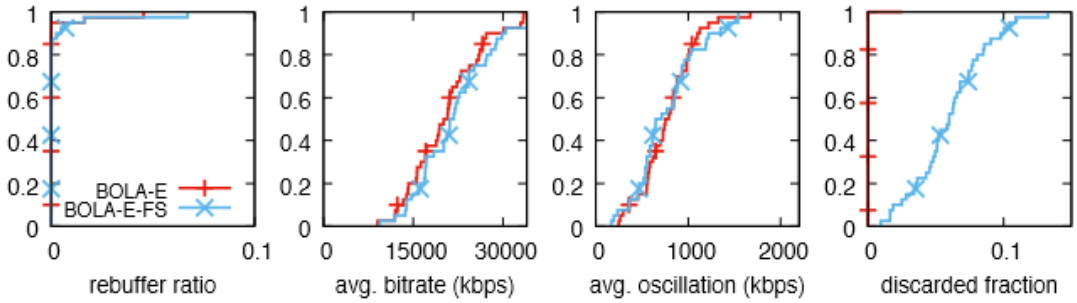
(b) Streaming the HD video.

Gambar 7.5 CDF metrik QoE dengan dan tanpa FAST SWITCHING menggunakan buffer 25 detik.

Gambar 7.5 juga menunjukkan bahwa untuk eksperimen di atas, untuk semua kasus FAST SWITCHING tidak secara nyata meningkatkan rebuffering dan untuk kebanyakan kasus tidak secara signifikan meningkatkan osilasi bitrate. Ini hanya meningkatkan osilasi bitrate secara signifikan untuk beberapa tes DINAMIS untuk video HD. Secara khusus, pada persentil ke-90, ini meningkatkan osilasi bitrate sebesar 306 kbps.

Perhatikan bahwa sementara metrik QoE untuk BOLA-E dan DYNAMIC serupa, ada perbedaan nyata dalam osilasi bitrate pada Gambar 7.5 (b). FAST SWITCHING menyebabkan level buffer turun, mengarahkan DYNAMIC untuk beralih ke algoritma THROUGHPUT. Karena jejak jaringan memiliki variabilitas bandwidth yang tinggi, algoritma THROUGHPUT memberikan osilasi bitrate yang lebih tinggi. Masalah ini tidak memengaruhi BOLA-E.

Kelemahan dari FAST SWITCHING adalah download overhead. Saat segmen pengganti diunduh, segmen yang diunduh sebelumnya dibuang dan tidak pernah diputar ulang ke penampil. Jadi biaya tambahan seperti itu menyebabkan biaya bandwidth tambahan. Untuk mengukur overhead, penulis menguji FAST SWITCHING dengan jejak 4G; jejak 4G memiliki variasi throughput yang terjadi pada frekuensi yang tipikal dalam skenario dunia nyata. Gambar 7.6 menunjukkan pecahan yang dibuang; pecahan yang dibuang adalah jumlah bit yang dibuang sebagai pecahan dari total bit yang diunduh untuk satu sesi. Median fraksi yang dibuang adalah 6%. Manfaat yang diperoleh dari overhead ini adalah waktu reaksi yang lebih cepat dan peningkatan bitrate rata-rata, dengan peningkatan bitrate sebesar 5% pada kasus median.

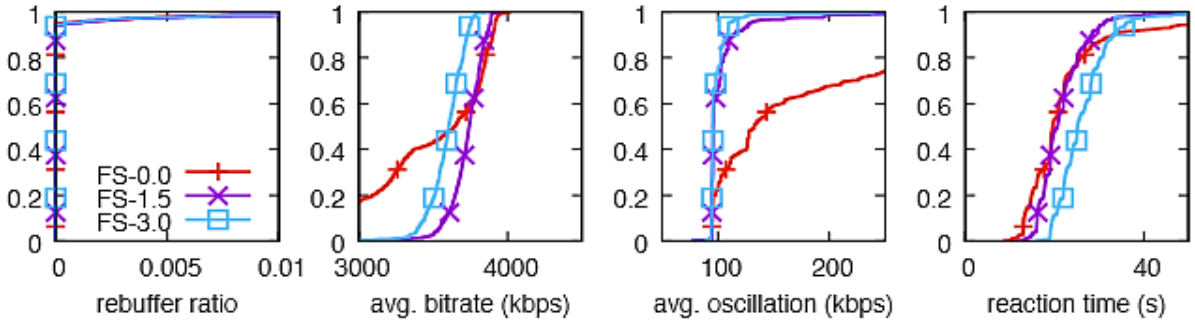


Gambar 7.6 CDF metrik QoE dengan FAST SWITCHING menggunakan jejak 4G. Fraksi yang dibuang adalah fraksi dari total bit yang diunduh yang diganti dan karenanya tidak pernah diputar ulang ke penampil.

7.2 DESAIN DASAR DALAM FAST SWITCHING

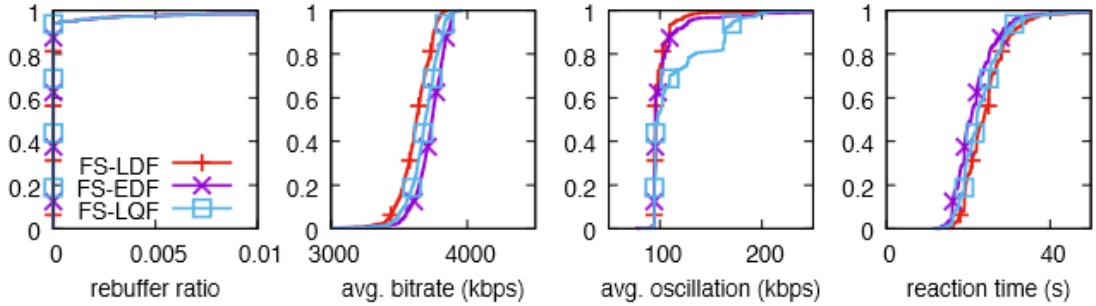
Gambar. 7.4–7.5 menunjukkan bahwa menggunakan FAST SWITCHING meningkatkan waktu reaksi terhadap peristiwa jaringan dan akibatnya juga meningkatkan metrik QoE. Pada bagian ini, penulis menguraikan dan mengevaluasi berbagai pilihan desain yang mungkin dilakukan dalam konteks penggantian segmen dan menjustifikasi pilihan yang dibuat dalam FAST SWITCHING.

1) Penggantian offset. FAST SWITCHING menggantikan segmen dengan mode early-deadline-first (EDF), dimulai dengan segmen yang paling dekat dengan play head. Namun, dimulai dengan segmen yang "terlalu dekat" dengan play head menimbulkan risiko segmen tersebut dimainkan sebelum pengunduhan pengganti selesai. Oleh karena itu, penulis menggunakan time offset untuk menentukan jarak (dalam detik) dari play head yang harus dipertimbangkan agar segmen dapat diganti dengan aman. Penulis bereksperimen dengan nilai yang berbeda untuk offset termasuk 0×, 1,5×, dan 3 × (panjang segmen) menggunakan jejak FCC yang meningkat dari throughput rendah ke throughput tinggi. Offset 0× berarti segmen pertama setelah segmen yang sedang diputar diganti. Gambar 7.7 menunjukkan metrik QoE untuk berbagai faktor saat menggunakan BOLA-E dengan FAST SWITCHING untuk video SD. Offset 0× menyebabkan osilasi berlebihan dan menyebabkan penurunan yang signifikan pada kecepatan bit rata-rata untuk separuh jejak. Di sisi lain, offset 3x lambat bereaksi dan tidak mendapatkan peningkatan bitrate rata-rata penuh dari penggantian. Penulis juga menjalankan pengujian untuk video lain seperti video HD, dan algoritme ABR lainnya seperti DYNAMIC dan offset pengganti lainnya antara 0× dan 3×. Penulis menemukan bahwa offset 1,5× adalah sweet spot. Ini adalah faktor terendah yang secara konsisten memiliki osilasi serendah offset 3x, tetapi memiliki waktu reaksi lebih cepat dan bitrate rata-rata lebih tinggi. Oleh karena itu, penulis memilih offset pengganti 1,5× dalam implementasi produksi FAST SWITCHING penulis.



Gambar 7.7. CDF metrik QoE BOLA-E dengan FAST SWITCHING dengan offset pengganti yang berbeda.

2) Urutan penggantian. Selain EDF, penulis mengeksplorasi cara lain untuk memesan segmen yang perlu diganti. Secara khusus, penulis mengevaluasi urutan last-deadline-first (LDF) di mana penggantian dimulai dari segmen yang terjauh dari play head. Penulis juga mengevaluasi kualitas terendah pertama (LQF) di mana segmen kualitas terendah dalam buffer diganti terlebih dahulu, meskipun tidak ada segmen yang berada dalam offset pengganti 1,5 × (panjang segmen) yang dapat diganti. Ketika ada beberapa kandidat LQF dengan bitrate rendah yang sama, penulis mengganti segmen yang akan memberikan metrik osilasi terendah setelah penggantian. Penulis mengevaluasi dan membandingkan EDF, LDF, dan LQF secara empiris, sekali lagi dengan jejak jaringan FCC. Gambar 7.8 menunjukkan metrik QoE untuk pesanan penggantian yang berbeda saat menggunakan BOLA-E dengan FAST SWITCHING untuk video SD. EDF bereaksi lebih cepat daripada LDF dan LQF. Misalnya, waktu reaksi median untuk EDF adalah 21 detik (resp., 33 detik), sedangkan LDF membutuhkan waktu 24 detik (resp., 42 detik) dan LQF membutuhkan waktu 22 detik (resp., 36 detik) saat memutar video SD (resp., HD). Akibatnya, penulis memilih EDF untuk implementasi produksi dash.js penulis.



Gambar 7.8. CDF metrik QoE BOLA-E dengan FAST SWITCHING untuk pesanan penggantian yang berbeda.

PEKERJAAN TERKAIT

Penulis telah menjelajahi pekerjaan terkait dalam algoritme ABR. Mengenai alat evaluasi, MACI menyediakan lingkungan emulasi yang memungkinkan pemuatan otomatis dan evaluasi pemain lengkap seperti dash.js. MACI dan Sabre melayani peran pelengkap

dalam pengembangan algoritma ABR. MACI adalah lingkungan lengkap yang memutar video, sedangkan Sabre adalah lingkungan simulasi yang dapat lebih cepat menguji algoritme ABR untuk berbagai video dan jejak jaringan tanpa benar-benar memutar video.

KESIMPULAN

Dalam buku ini merancang dan mengimplementasikan Sabre, alat sumber terbuka yang tersedia untuk umum yang dapat digunakan oleh peneliti untuk menjalankan simulasi algoritme ABR yang akurat menggunakan arsitektur pemutar yang mirip dengan dash.js. Penulis menggunakan Sabre untuk merancang dan menguji BOLA-E dan DYNAMIC, dua algoritme yang menyempurnakan algoritme BOLA berbasis buffer ABR. Penulis juga mengembangkan algoritme FAST SWITCHING yang dapat menggantikan segmen yang telah diunduh dengan segmen dengan kecepatan bit lebih tinggi (sehingga berkualitas lebih tinggi). Algoritme baru memberikan QoE yang lebih tinggi kepada pengguna dalam hal bitrate yang lebih tinggi, rebuffer yang lebih sedikit, dan osilasi bitrate yang lebih rendah. Selain itu, algoritme ini bereaksi lebih cepat terhadap kejadian pengguna seperti memulai dan mencari, dan merespons lebih cepat terhadap kejadian jaringan seperti peningkatan throughput. Selain itu, mereka bekerja dengan baik untuk streaming langsung yang memerlukan latensi rendah, masalah yang menantang untuk algoritme ABR, karena buffer klien harus dijaga sangat kecil. Secara keseluruhan, algoritme yang disajikan dalam bab penulis menawarkan QoE video yang unggul dan daya tanggap untuk streaming video adaptif kehidupan nyata. Ketiga algoritme yang disajikan dalam bab ini sekarang menjadi bagian dari dash.js pemutar referensi DASH resmi dan sedang digunakan oleh penyedia video di lingkungan produksi. Melalui dash.js, BOLA kini digunakan dalam produksi oleh beberapa penyedia video besar dan jaringan pengiriman seperti Akamai, BBC, CBS, dan Orange. Untuk penyedia video yang ingin memilih BOLA-E versus DYNAMIC di pemutar produksi, penulis juga membandingkannya secara langsung. BOLA-E sedikit lebih baik daripada DYNAMIC ketika bandwidth jaringan memiliki variabilitas yang lebih tinggi, situasi yang juga sangat menantang untuk algoritma THROUGHPUT. Di sisi lain, DYNAMIC sedikit lebih baik untuk kapasitas buffer kecil, situasi yang dapat menjadi tantangan untuk semua algoritme berbasis buffer termasuk BOLA-E. Meskipun DYNAMIC saat ini merupakan pilihan default, penulis menemukan bahwa kedua algoritme memiliki kinerja yang sama baiknya dalam metrik QoE dan daya tanggap.

BAB 8
STREAMING VIDEO ADAPTIF

Sementara sistem streaming video awal menggunakan Internet sebagai sistem upaya terbaik tanpa transportasi yang andal, video modern menggunakan HTTP adaptive streaming (HAS) sebagai metode pengiriman video default. Manfaat HTTP dan protokol TCP yang mendasarinya adalah bahwa sistem HAS memiliki transportasi yang andal. Namun, keandalan TCP ada harganya. Pengiriman ulang paket yang hilang menimbulkan penundaan dan dapat menyebabkan pemblokiran head-of-line.

Jalan tengah antara transportasi upaya terbaik dan transportasi yang andal adalah keandalan selektif seperti teknik yang diusulkan oleh Feamster et al., di mana beberapa frame dikirim melalui transportasi yang dapat diandalkan dan frame lainnya dikirim melalui transportasi yang tidak dapat diandalkan. Namun, TCP tidak mendukung metode seperti itu.

QUIC adalah alternatif dari TCP untuk HAS. Streaming video menggunakan QUIC dapat memiliki beberapa manfaat kinerja. QUIC tidak memiliki jabat tangan SYN tiga arah dan membutuhkan lebih sedikit perjalanan bolak-balik untuk membangun koneksi yang aman, memberikan waktu pengaktifan video yang lebih cepat. Namun, transportasi andal QUIC masih memiliki masalah yang sama dengan TCP.

Penulis mengusulkan VOXEL, sistem streaming video baru yang menggabungkan sebagian transportasi yang andal dengan HAS. Meskipun penulis membangun ide sebelumnya, sepengetahuan penulis VOXEL adalah sistem pertama yang menggabungkan transportasi yang andal dan tidak andal dengan streaming kecepatan bit adaptif (ABR) melalui HTTP. VOXEL dapat mentolerir dua jenis kerugian. Pertama, memungkinkan beberapa paket hilang dalam frame non-key, mirip dengan. Kedua, itu benar-benar dapat menjatuhkan beberapa bingkai dalam suatu segmen. Kedua jenis kerugian tersebut dapat diizinkan untuk meningkatkan pengalaman secara keseluruhan, sebagian besar dengan menghindari peristiwa penyanggaan ulang. VOXEL juga memungkinkan algoritme ABR untuk mengeksploitasi keandalan selektif. Kontribusi utama penulis adalah perluasan BOLA-E yang mengeksploitasi kemampuan transportasi baru untuk meningkatkan QoE.

8.1 VOXEL: pertimbangan sistem

Ada beberapa solusi streaming video yang menyetel protokol transport atau mengoptimalkan lapisan aplikasi, yaitu algoritme ABR. Dengan VOXEL, penulis mengusulkan pendekatan pengoptimalan lintas lapisan. Penulis sekarang menjelaskan beberapa ide kunci di balik pengembangan VOXEL.

Pengiriman video lossy masih bisa mencapai QoE yang tinggi

Konten video dapat menyebabkan hilangnya beberapa frame tanpa dampak signifikan pada QoE. Selanjutnya, hukuman QoE dari menjatuhkan sebuah frame tergantung pada kepentingan relatif dari frame di dalam semgent daripada pada ukuran frame. Ketika sebuah frame dijatuhkan, penurunan QoE keseluruhan tergantung pada berapa banyak frame lain di

segmen tersebut yang memiliki referensi ke frame yang dijatuhkan karena kesalahan menyebar melalui referensi tersebut. Perhatikan bahwa kunci I-Frame di setiap segmen tidak bergantung pada bingkai lainnya, dan sebagian besar bingkai lainnya bergantung secara langsung atau tidak langsung pada I-Frame kecuali jika ada perubahan pemandangan di segmen tersebut.

Konten video juga dapat mentolerir kehilangan paket dalam bingkai, di mana hanya sebagian kecil dari bingkai yang hilang. Sementara informasi header dapat membuat seluruh frame tidak dapat digunakan, kehilangan informasi non-header dapat ditoleransi.

Hubungan kompleks antara bitrate dan QoE

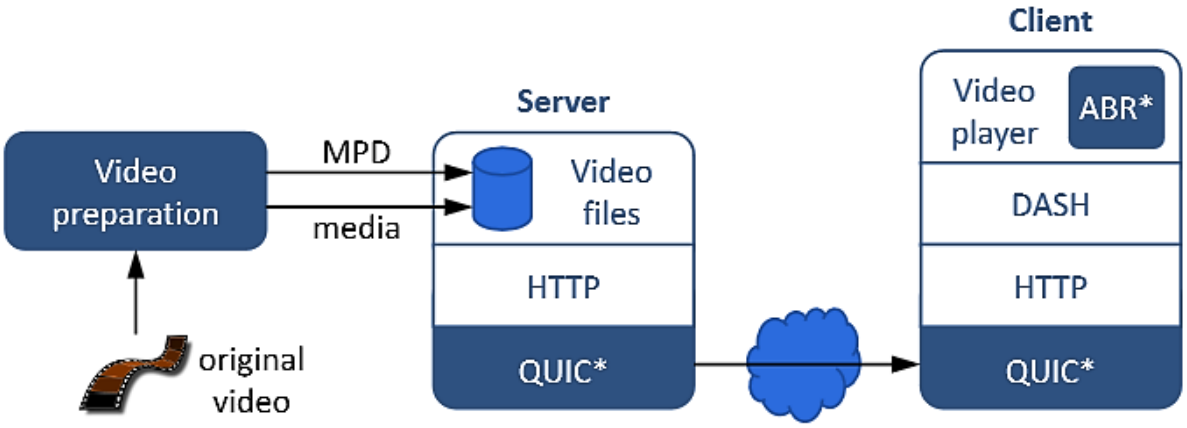
Algoritme ABR modern berusaha untuk mengoptimalkan QoE secara tidak langsung dengan mengoptimalkan metrik seperti bitrate. Algoritme umumnya menggunakan beberapa fungsi bitrate seperti fungsi logaritmik pada (2.21) untuk memberikan hasil yang semakin berkurang ketika meningkatkan bitrate. Namun, hubungan antara metrik seperti bitrate dan QoE adalah kompleks dan fungsi semacam itu hanya mendekati QoE. Saat menjelajahi pengiriman video lossy, metrik seperti bitrate tidak dapat menangkap kepentingan relatif dari frame yang berbeda dalam segmen karena ukuran frame dalam bit tidak menangkap informasi seperti ketergantungan antar-frame.

Tiga metrik QoE yang umum adalah rasio sinyal terhadap noise puncak (PSNR), kesamaan struktural (SSIM) dan video multimethod assesment fusion (VMAF). Penulis menggunakan SSIM untuk diskusi penulis karena memiliki korelasi yang lebih baik dengan skor opini rata-rata (MOS) daripada PSNR. VMAF memiliki korelasi yang lebih baik daripada SSIM dalam hal kompresi dan artefak penskalaan, tetapi VMAF tidak mendukung kehilangan paket. Romaniak dkk menunjukkan bahwa SSIM dapat menangkap artefak yang disebabkan oleh kehilangan paket atau penurunan bingkai.

Menjatuhkan bingkai dapat menjembatani kesenjangan antara tingkat bitrate

Algoritme ABR mendukung peralihan antara kecepatan bit yang berbeda pada batas segmen. Namun, ada beberapa kelemahan memiliki terlalu banyak tingkat bitrate. Pertama, memiliki lebih banyak tingkat bitrate memerlukan lebih banyak transcoding pada fase persiapan konten dan, yang lebih penting, menggunakan lebih banyak penyimpanan. Kedua, memiliki tingkat bitrate yang lebih tinggi akan menurunkan kemungkinan hit cache karena kemungkinan besar klien yang berbeda melakukan streaming pada kecepatan bit yang berbeda.

Salah satu cara untuk menjembatani kesenjangan antara dua kecepatan bit adalah dengan membuang beberapa frame yang kurang penting dari segmen pada kecepatan bit yang lebih tinggi. Meskipun algoritme ABR biasanya tidak mendownload segmen pada bitrate yang lebih tinggi dari estimasi throughput jaringan, mengizinkan penurunan frame memberi klien lebih banyak fleksibilitas untuk menggunakan lebih banyak bandwidth yang tersedia. Selain itu, klien dapat memutuskan untuk melepaskan beberapa bingkai dari segmen saat segmen sedang dalam proses pengunduhan. Dengan demikian, klien dapat mengunduh pada bitrate yang lebih tinggi dengan mengetahui bahwa jika kondisi jaringan memburuk, rebuffering dapat dihindari dengan menjatuhkan beberapa frame.

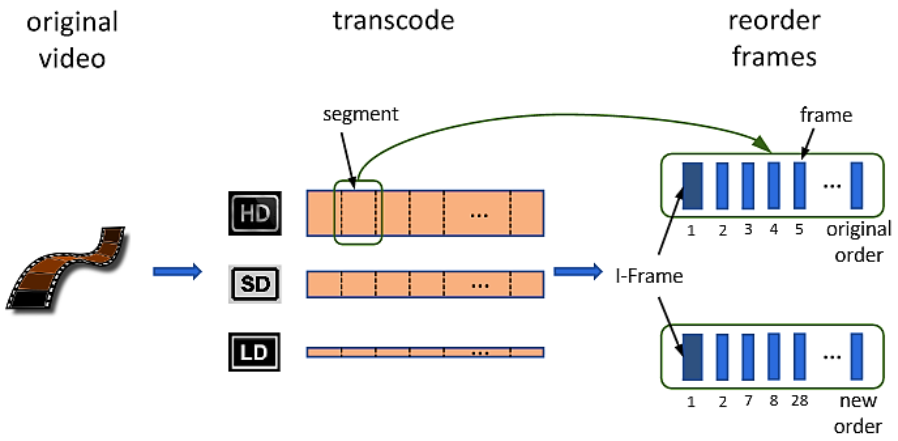


Gambar 8.1. Streaming video menggunakan VOXEL. VOXEL berbeda dari streaming tradisional dalam tiga area: persiapan video, lapisan transport QUIC*, dan algoritme adaptasi ABR*.

8.2 VOXEL: arsitektur sistem

Buku ini sekarang menjelaskan VOXEL, sistem streaming video yang penulis kembangkan berdasarkan gagasan di gambar 4.1. Gambar 4.1 menunjukkan komponen utama VOXEL. Meskipun mirip dengan sistem tradisional, VOXEL memiliki tiga perbedaan penting.

1. Penulis menyiapkan konten video untuk streaming dan mengidentifikasi kepentingan relatif dari frame di setiap segmen video. Penulis mengatur ulang bingkai di setiap segmen sesuai urutan kepentingannya.
2. Pada lapisan transport, penulis membangun ekstensi QUIC untuk menawarkan sebagian transportasi yang andal, memungkinkan penulis untuk mengeksploitasi pengamatan bahwa tidak semua frame memerlukan pengiriman yang andal.
3. Pada lapisan aplikasi, penulis merancang algoritme ABR yang menggunakan informasi dari tahap persiapan video dan berinteraksi erat dengan lapisan transport untuk mencapai QoE yang lebih tinggi.



Gambar 8.2. Mempersiapkan konten video untuk VOXEL.

VOXEL kompatibel dengan klien yang sudah ada. Klien yang tidak mengetahui VOXEL masih dapat melakukan streaming video dari server VOXEL. Klien seperti itu akan meminta

aliran melalui transportasi tradisional yang andal. Selain itu, klien yang sadar VOXEL dapat melakukan streaming video menggunakan algoritme ABR yang tidak sadar VOXEL. Klien seperti itu akan menerima sebagian tetapi tidak semua manfaat VOXEL. Dukungan untuk berbagai tingkat kesadaran VOXEL memudahkan penerapan VOXEL secara bertahap di Internet.

Mempersiapkan konten video

Gambar 8.2 menunjukkan bagaimana VOXEL menyiapkan konten video. Langkah pertama adalah mentranskode video, mirip dengan solusi HAS yang paling umum. Penulis mentranskode video asli ke sejumlah bitrate berbeda dan mempartisi setiap video yang ditranskode menjadi segmen yang dapat disajikan menggunakan manifes DASH biasa. Penulis kemudian menambahkan langkah ekstra untuk memprioritaskan frame dalam setiap segmen. Hal ini memungkinkan klien untuk mendownload frame sesuai urutan pentingnya dan meningkatkan QoE yang diperoleh oleh segmen yang diunduh sebagian saat download segmen terganggu.

Untuk setiap segmen, penulis mencoba tiga urutan prioritas yang berbeda. I-Frame memiliki kepentingan tertinggi di semua pesanan dan selalu diunduh terlebih dahulu.

1. Urutan asli di mana penulis mengirimkan segmen dalam urutan decoding reguler (MPEG).
2. Urutkan berdasarkan referensi masuk di mana penulis memberi peringkat bingkai menggunakan jumlah referensi masuk dari bingkai lain. Penulis kemudian menyusun ulang bingkai, dan bingkai dengan referensi masuk yang lebih sedikit dipindahkan ke akhir segmen. Penulis menghitung referensi masuk langsung dan tidak langsung karena ketergantungan bingkai dapat disebarkan melalui referensi tidak langsung.
3. Urutkan dengan mengelompokkan frame yang tidak direferensikan di mana kita memindahkan frame yang tidak memiliki referensi masuk dari frame lain ke akhir segmen. Penulis meninggalkan bingkai dengan referensi masuk dalam urutan asli, dan penulis juga mempertahankan urutan relatif bingkai tanpa referensi masuk. Urutan ini merupakan langkah perantara antara dua urutan pertama.

Penulis memilih pesanan dari ketiganya untuk setiap segmen secara terpisah. Untuk memilih pesanan, penulis menganalisis penurunan QoE saat mengunduh sebagian segmen untuk masing-masing dari tiga pesanan. Untuk setiap pesanan, penulis membuang bingkai di segmen mulai dari akhir, dan penulis menghitung skor SSIM untuk setiap iterasi. Proses ini memberi kita tabel yang memetakan jumlah bit yang diunduh ke skor SSIM untuk setiap pesanan untuk setiap segmen. Saat menganalisis segmen pada tingkat bitrate m, penulis menggunakan skor SSIM untuk segmen lengkap pada tingkat bitrate berikutnya yang lebih rendah m − 1 sebagai referensi SSIM. Untuk setiap pesanan, penulis mencari ukuran unduhan parsial pada kualitas m yang mendapatkan referensi SSIM. Penulis memilih pesanan yang memiliki ukuran terkecil.

Penulis menyertakan urutan bingkai beserta tabel ukuran dan skor SSIM terkait sebagai metadata dalam manifes DASH. Untuk metadata tambahan meningkatkan ukuran file manifes dengan faktor sekitar 8.

Namun, ukuran file manifes masih jauh lebih kecil dari ukuran segmen media. Klien yang memahami VOXEL dapat mengeksploitasi metadata ini dan mengunduh bingkai yang

diperlukan sesuai urutan pentingnya. Klien yang tidak mengetahui VOXEL dapat dengan mudah mengabaikan meta-data ini dan mengunduh segmen dalam urutan pendekodean biasa.

QUIC*: Transport Layer

Lapisan transport VOXEL dibangun di atas ekstensi QUIC. Sementara aliran kontrol terpisah. Dalam buku ini menggunakan header HTTP untuk mengontrol aliran media. Penulis merancang versi QUIC* yang dimodifikasi, yang penulis sebut QUIC*. QUIC* mendukung streaming yang andal di vanilla QUIC* dan juga mendukung streaming yang tidak dapat diandalkan tanpa transmisi ulang. Tidak seperti UDP mentah, aliran yang tidak dapat diandalkan di QUIC* masih menggunakan mekanisme kemacetan dan kontrol aliran dari koneksi QUIC. Klien dapat membuka aliran yang andal hanya dengan mengirimkan permintaan HTTP GET. Klien yang ingin membuka aliran yang tidak dapat diandalkan mengirimkan permintaan HTTP GET tetapi menyertakan tajuk HTTP khusus x-voxel-tidak dapat diandalkan. Klien yang tidak mengetahui VOXEL tidak akan menggunakan tajuk khusus dan hanya akan menggunakan aliran yang andal. Kompatibilitas mundur ini memungkinkan penyebaran bertahap.

Selama sesi video, klien VOXEL menggunakan dua permintaan HTTP per segmen. Permintaan pertama mendapatkan informasi I-Frame dan header untuk semua frame melalui aliran yang andal. Permintaan kedua mendapatkan data video untuk frame yang tersisa melalui aliran yang tidak dapat diandalkan. Paket yang hilang pada aliran yang tidak dapat diandalkan tidak ditransmisikan ulang, menyebabkan penurunan QoE. Perhatikan bahwa bingkai biasanya menjangkau beberapa paket, dan kehilangan paket tidak diterjemahkan langsung ke bingkai yang dijatuhkan tetapi bingkai yang rusak sebagian. Sementara transportasi yang tidak dapat diandalkan dapat menyebabkan hilangnya paket, itu juga bermanfaat untuk menghindari pemblokiran head-of-line dan gangguan aliran. Karena header dikirimkan dengan andal, VOXEL dapat menjaga integritas struktur data dengan mengisi lubang dengan padding nol. Hal ini menyebabkan artefak yang terkait dengan kehilangan data, tetapi pemutar video masih dapat menggunakan sisa data bingkai. Eksperimen dalam buku ini menunjukkan bahwa VOXEL secara signifikan mengurangi buffering ulang, terutama dalam skenario dengan ukuran buffer yang lebih kecil yang biasa terjadi dalam skenario seperti streaming langsung waktu nyata.

VOXEL juga mengeksploitasi perilaku klien video tipikal untuk mentransmisikan ulang data yang hilang secara selektif pada aliran yang tidak dapat diandalkan. Klien video memiliki periode dengan jeda dalam proses pengunduhan. Jeda seperti itu dapat terjadi ketika level buffer berada pada kapasitas maksimum. VOXEL menggunakan periode tersebut untuk mengirim ulang paket yang hilang pada aliran yang tidak dapat diandalkan.

Menyempurnakan Algoritma ABR

VOXEL menyediakan kerangka kerja untuk algoritme ABR kelas baru dengan fitur utama berikut. Optimalkan untuk QoE. Algoritme ABR seperti BOLA mengoptimalkan fungsi utilitas yang biasanya berdasarkan bitrate. Algoritme semacam itu biasanya memungkinkan fungsi utilitas yang berbeda. VOXEL menggunakan utilitas berbasis metrik QoE. Penulis

menggunakan SSIM dalam evaluasi penulis, tetapi VOXEL adalah agnostik QoE-metrik. Hasil eksperimen digeneralisasikan ke metrik lain seperti VMAF dan PSNR.

Mendukung unduhan sebagian-segmen. Algoritme ABR tradisional memilih kecepatan bit pada batas segmen. Mereka dapat memilih dari serangkaian bitrate terbatas. VOXEL memungkinkan pengunduhan segmen parsial dan menyajikan metrik QoE masing-masing untuk subset pengunduhan yang berbeda, sehingga secara signifikan meningkatkan ruang keputusan yang tersedia.

Opsi pengabaian segmen. Algoritme ABR tradisional mungkin mengabaikan segmen saat mendownload segmen. Mereka meninggalkan pengunduhan segmen dengan kecepatan bit tinggi untuk memulai pengunduhan dengan kecepatan bit rendah jika mereka mendeteksi risiko buffering ulang yang tinggi. VOXEL memperkenalkan opsi lain: hentikan pengunduhan, tetapi pertahankan sebagian segmen dan lanjutkan ke yang berikutnya. Opsi baru mengunduh bit lebih sedikit daripada algoritme ABR tradisional selama periode dengan kondisi jaringan yang buruk. Juga, segmen bitrate tinggi parsial mungkin memberikan QoE yang lebih baik daripada segmen bitrate rendah penuh.

Untuk melengkapi implementasi VOXEL penulis, penulis mengembangkan ABR*, sebuah algoritma ABR baru berdasarkan BOLA-E. BOLA-E sudah mendukung skenario buffer rendah, dan tes pendahuluan dengan BOLA-E yang tidak dimodifikasi pada QUIC* menunjukkan bahwa BOLA-E bekerja sedikit lebih baik pada QUIC* bila dibandingkan dengan BOLA-E pada QUIC. Hal ini menjadikan BOLA-E algoritme dasar yang baik untuk membangun ABR*.

ABR*: Memperluas BOLA-E untuk VOXEL

Buku ini mengembangkan BOLA-E yang mendukung skenario buffer rendah. Namun, seperti halnya algoritme ABR lainnya, penurunan kondisi jaringan akan menyebabkan BOLA-E menurunkan bitrate dan akhirnya melakukan buffer ulang. Dengan mengubah lapisan transport dari QUIC ke QUIC*, kita dapat menghindari beberapa transmisi ulang paket. Hal ini mengurangi interupsi aliran dengan mengorbankan beberapa kerusakan video terbatas, dengan peningkatan QoE secara keseluruhan. Namun, pengoptimalan lintas lapisan dapat memungkinkan peningkatan yang lebih baik. Penulis sekarang menyajikan serangkaian peningkatan pada BOLA-E yang mengarah ke ABR*-O, algoritme ABR yang mengeksploitasi sistem VOXEL lainnya untuk lebih meningkatkan QoE.

BAB 9
METODOLOGI EKSPERIMENTAL

Untuk mengevaluasi algoritma, penulis menggunakan satu set 86 jejak 3G. Penulis menggunakan jejak 3G karena dalam buku ini ingin menjelajahi VOXEL dengan kondisi jaringan yang buruk. Meskipun penulis berharap sebagian besar algoritme ABR bekerja dengan baik dengan kondisi jaringan yang baik, penulis ingin merancang algoritme ABR yang membutuhkan waktu lebih lama untuk mengalami degradasi QoE karena kondisi jaringan memburuk.

Karena Sabre tidak mendukung simulasi lapisan transport, dalam percobaan ini menguji algoritme penulis pada mesin fisik. Setiap sesi simulasi video dijalankan pada tiga mesin bare-metal yang menjalankan Linux: satu mesin mengemulasi server, satu mesin mengemulasi router perantara, dan satu mesin mengemulasi mesin klien. Membentuk arus lalu lintas melalui router menggunakan utilitas kontrol lalu lintas Linux tc, mengubah bandwidth setiap detik agar sesuai dengan jejak jaringan yang digunakan. Penulis menguji algoritma ABR menggunakan empat video berbeda: *Big Buck Bunny* (BBB), *Elephants Dream* (ED), *Sintel*, dan *Tears of Steel* (ToS). Dalam buku ini mentranskode setiap video pada 14 bitrate, dengan resolusi mulai dari 144p hingga 2160p dan bitrate mulai dari 160 kbps hingga 10 Mbps. Buku ini mempartisi video dalam segmen 4 detik, dan memilih kutipan 5 menit dari setiap video.

9.1 BOLA-SSIM: Memasukkan utilitas SSIM dan unduhan parsial

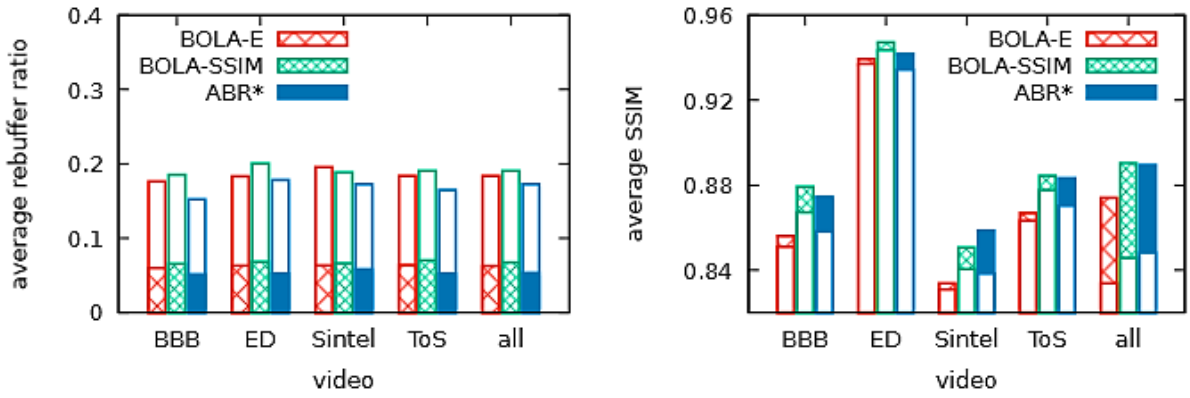
Fase persiapan konten VOXEL menyediakan setiap segmen pada beberapa kecepatan bit yang serupa dengan sistem HAS tradisional. VOXEL juga memberikan tabel cut-off point di setiap segmen dengan skor SSIM untuk setiap cut-off point. Namun, BOLA-E hanya mempertimbangkan untuk mengunduh segmen pada setiap kecepatan bit. Penulis pertama kali memperbarui BOLA-E untuk memperluas ruang keputusan. Setiap representasi kecepatan bit dapat diunduh seluruhnya atau diunduh hingga salah satu titik batas. Karena penulis mencoba mengeksploitasi kepentingan relatif yang berbeda dari bingkai, fungsi utilitas berbasis ukuran tidak lagi berfungsi. Karenanya, penulis memperbarui BOLA-E untuk menyetel fungsi utilitas ke skor SSIM. Penulis memanggil algoritme setelah dua pembaruan BOLA-SSIM.

Evaluasi

Percobaan ini menjalankan sesi video dengan BOLA-E dan BOLA-SSIM, keduanya dengan kapasitas buffer 8 detik, melalui 86 jejak 3G menggunakan masing-masing dari empat video. Perhatikan bahwa dengan kapasitas penyangga 8 detik, pemain tidak mulai mengunduh segmen 4 detik baru jika tingkat penyangga di atas 4 detik. Pada gambar yang disajikan di Gambar 4.3 menunjukkan rasio rebuffer rata-rata (waktu yang dihabiskan rebuffering dibagi dengan total waktu) dan skor SSIM untuk sesi tersebut. BOLA-SSIM memperoleh skor SSIM yang lebih tinggi di semua video jika dibandingkan dengan BOLA. Namun, ini harus dibayar dengan sedikit peningkatan dalam rebuffering. BOLA-SSIM memiliki keunggulan optimalisasi

langsung untuk SSIM. Selain itu, opsi pengunduhan yang ditingkatkan memungkinkan BOLA-SSIM menjadi lebih agresif. Jika estimasi throughput yang berkelanjutan berada di antara dua bitrate, BOLA-E mungkin mengunduh pada bitrate yang lebih rendah dari keduanya. Di sisi lain, BOLA-SSIM dapat memilih bitrate yang lebih tinggi dengan satu atau lebih frame yang dijatuhkan. Sementara BOLA-SSIM secara agresif menggunakan lebih banyak bandwidth, ia memiliki risiko buffer ulang yang lebih tinggi.

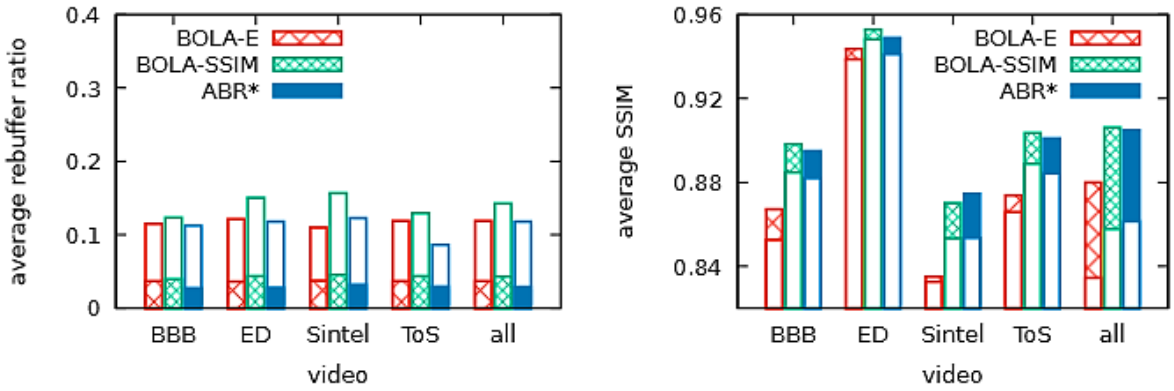
Percobaan ini mengulangi percobaan dengan kapasitas buffer 32 detik, memplot hasilnya pada Gambar 9.1. Sekali lagi, BOLA-SSIM memperoleh skor SSIM yang lebih tinggi dengan sedikit peningkatan dalam buffering ulang. Perhatikan bahwa buffer yang lebih besar memberikan rebuffering yang lebih rendah dan SSIM yang lebih tinggi di semua video dan semua algoritme seperti yang diharapkan.



Gambar 9.1. Rasio buffer ulang rata-rata dan SSIM rata-rata lebih dari 86 jejak 3G untuk BOLA-E, BOLA-SSIM dan ABR* dengan kapasitas buffer 8 detik untuk empat video secara individu dan bersama-sama (semua). Plot rasio penyangga menampilkan rata-rata (pola) dan persentil ke-90 (polos). Plot SSIM menampilkan mean (pola) dan persentil ke-10 (polos).

9.2 ABR*: PENINGKATAN PENGABAIAN UNDUHAN

BOLA-SSIM menggunakan unduhan segmen parsial yang disediakan oleh VOXEL untuk memilih unduhan baru. Penulis juga dapat menggunakan unduhan segmen parsial sambil meninggalkan segmen selama unduhan. Ingatlah bahwa pengabaian pengunduhan terjadi saat pengunduhan memakan waktu terlalu lama dan berisiko tinggi melakukan buffer ulang sebelum selesai. Sementara BOLA-E dan BOLA-SSIM dapat mengabaikan pengunduhan untuk mengunduh kecepatan bit yang lebih rendah, VOXEL menyediakan opsi lain. Bagian unduhan yang sudah selesai dapat digunakan sebagai segmen parsial. Dengan demikian, tidak perlu memulai ulang unduhan dengan bitrate yang lebih rendah. Dalam buku ini memanggil algoritme setelah pembaruan ini ABR*.

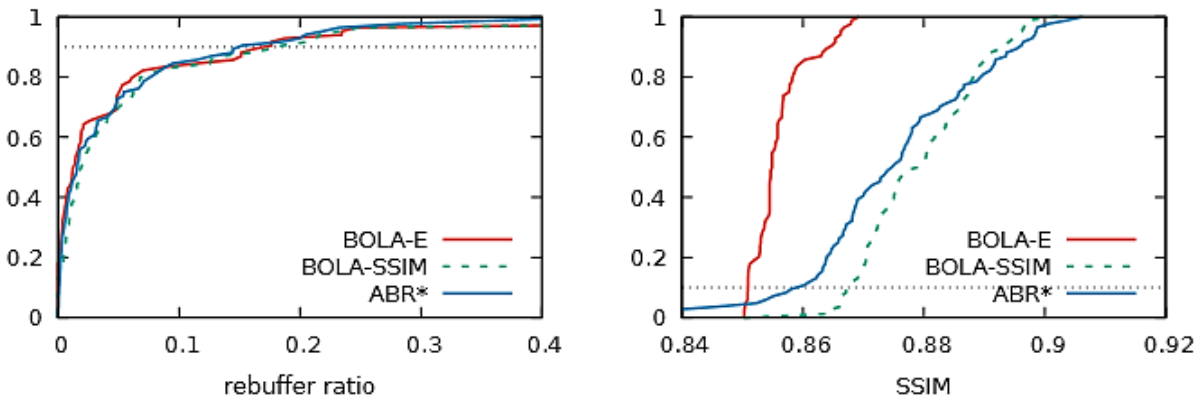


Gambar 9.2 Rasio buffer ulang rata-rata dan SSIM rata-rata lebih dari 86 jejak 3G untuk BOLA-E, BOLA-SSIM dan ABR* dengan kapasitas buffer 32 detik untuk empat video secara individu dan bersama-sama (semua). Plot rasio penyangga menampilkan rata-rata (pola) dan persentil ke-90 (polos). Plot SSIM menampilkan mean (pola) dan persentil ke-10 (polos).

Evaluasi

Meskipun ABR* masih memperoleh skor SSIM yang lebih tinggi daripada BOLA-E, ABR* memperoleh skor SSIM yang sedikit lebih rendah daripada BOLA-SSIM. Hal ini diharapkan karena ABR* mungkin memutuskan untuk menghentikan pengunduhan dan mendapatkan SSIM yang lebih rendah daripada menyelesaikan pengunduhan dan menyebabkan peristiwa penyanggaan ulang. Di sisi lain, ABR* memiliki rebuffering yang lebih rendah daripada BOLA-E dan BOLA-SSIM.

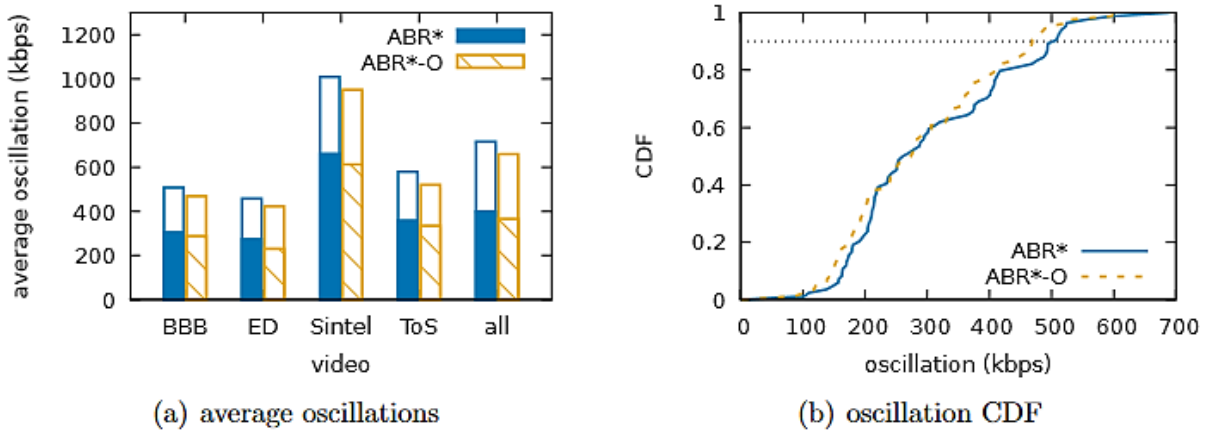
Gambar 9.2 menunjukkan CDF untuk rasio penyangga ulang dan SSIM yang sesuai dengan bar BBB pada Gambar 9.3. Perhatikan perilaku pada jejak yang paling menantang. Sementara skor SSIM untuk BOLA-E tidak di bawah 0,850, skor SSIM untuk ABR* adalah 0,792 untuk kasus terburuk. Namun, sementara rasio penyangga ulang untuk ABR* naik hingga maksimum 45,4%, BOLA-E naik menjadi 76,0%. Hal ini terjadi karena selama kondisi jaringan yang ekstrim, jumlah kasus saat ABR* dapat menghentikan pengunduhan dan mempertahankan segmen yang selesai sebagian meningkat. Perhatikan bahwa sementara sesi dengan rebuffering sebanyak itu tidak dapat ditonton, jejak jaringan terburuk masih dapat menunjukkan penurunan yang lebih pendek dalam kondisi jaringan yang dapat diterima.



Gambar 9.3 Rasio buffer ulang dan CDF SSIM lebih dari 86 jejak 3G untuk BOLA-E, BOLA-SSIM, dan ABR* dengan kapasitas buffer 8 detik untuk video BBB.

9.3 ABR*-O: Mengurangi osilasi bitrate

BOLA-E menghindari osilasi bitrate dengan memperluas BOLA-O. Ingatlah bahwa BOLA-O menghindari osilasi bitrate dengan tidak memilih bitrate yang lebih tinggi dari perkiraan bandwidth berkelanjutan. Namun, ABR* memiliki banyak opsi pengunduhan dan tindakan pengunduhan terdiri dari skor SSIM dari tabel skor yang tersedia untuk setiap segmen, bukan kecepatan bit. Dengan demikian, ABR* lebih sering beralih di antara kecepatan bit. Perhatikan bahwa untuk tujuan mengukur sakelar kecepatan bit, penulis tidak membedakan antara segmen tertentu yang diunduh seluruhnya atau sebagian.



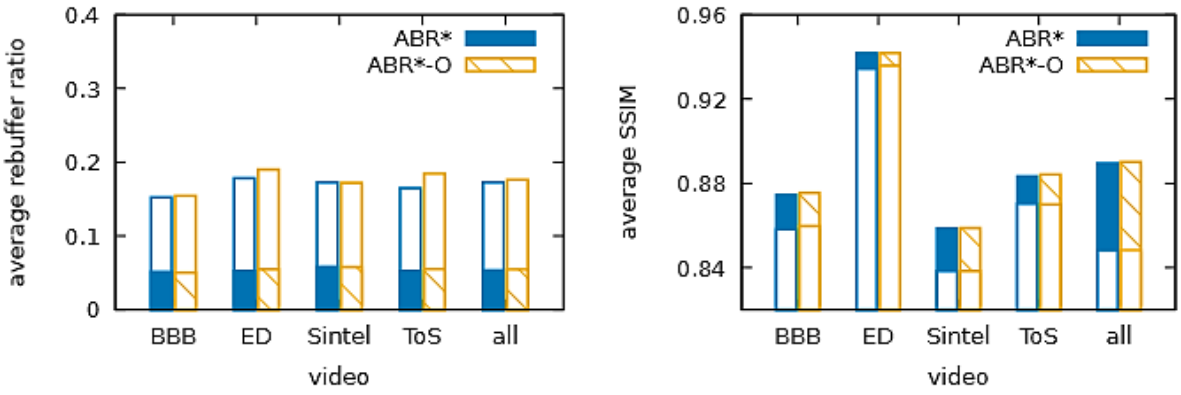
Gambar 9.4. Osilasi kecepatan bit pada 86 pelacakan 3G untuk ABR* dan ABR*-O dengan kapasitas buffer 8 detik. (a) Rata-rata osilasi bitrate untuk empat video secara individual dan bersama-sama (semua). Plot menampilkan rata-rata (pola) dan persentil ke-90 (polos). (b) CDF osilasi bitrate untuk video BBB.

Untuk mengurangi osilasi, penulis menghukum sakelar kecepatan bit apa pun. Saat ABR* memilih bitrate berikutnya, penulis memberikan penalti untuk setiap bitrate yang berbeda dari bitrate yang terakhir diunduh. Jika terdapat M tingkat bitrate, $\upsilon 1$ adalah skor SSIM untuk bitrate terendah, dan υM adalah skor SSIM untuk bitrate tertinggi, maka kita menetapkan pinalti perubahan bitrate sebagai $(\upsilon M - \upsilon 1)/(M - 1)$. Ini sesuai dengan perbedaan rata-rata utilitas antara tingkat bitrate yang berdekatan. Penalti membuat algoritme ABR* lebih enggan mengubah bitrate, tetapi masih akan mengubah bitrate jika perbedaan utilitas lebih tinggi daripada penalti. Penulis menyebut algoritme yang diperbarui ini ABR*-O.

Evaluasi

Buku ini mengulangi percobaan untuk ABR*-O menggunakan kapasitas buffer 8s untuk membandingkan ABR* dengan ABR*-O. Gambar 9.5 menunjukkan rata-rata osilasi bitrate untuk semua video dan CDF osilasi bitrate untuk video BBB. ABR*-O mencapai osilasi rata-rata 5,4% lebih sedikit, dan osilasi 7,6% lebih sedikit pada persentil ke-90. Penulis juga membandingkan rasio rebuffer rata-rata dan rata-rata SSIM pada Gambar 4.7 dan tidak

menemukan perbedaan yang signifikan. Dengan demikian, ABR*-O mengurangi osilasi bitrate tanpa degradasi apa pun dalam rebuffering atau SSIM.



Gambar 9.5. Rasio buffer ulang rata-rata dan SSIM rata-rata lebih dari 86 pelacakan 3G untuk ABR* dan ABR*-O dengan kapasitas buffer 8 detik untuk empat video secara terpisah dan bersama-sama (semuanya). Plot rasio penyangga menampilkan rata-rata (pola) dan persentil ke-90 (polos). Plot SSIM menampilkan mean (pola) dan persentil ke-10 (polos).

Pekerjaan terkait

Sistem streaming video awal menggunakan Internet sebagai sistem upaya terbaik tanpa transportasi yang andal. RTSP dan RTP biasanya digunakan masing-masing untuk pensinyalan dan pengiriman media. Meskipun pensinyalan RTSP dapat dikirimkan melalui TCP atau UDP, umumnya dikirimkan melalui TCP. RTP juga dapat dikirimkan pada TCP atau UDP. Namun, RTP sebagian besar disampaikan melalui UDP. Ekspansi transmisi ulang selektif ke RTP disebut SR-RTP mengeksplorasi jalan tengah antara menggunakan protokol yang dapat diandalkan atau tidak dapat diandalkan. SR-RTP mentransmisikan ulang paket yang dijatuhkan di I-Frame kunci tetapi tidak mentransmisikan ulang paket yang dijatuhkan di bingkai lain.

Aplikasi awal untuk streaming video menyertakan beberapa opsi komersial. Solusi RealVideo RealNetworks menggunakan UDP yang kuat untuk mengurangi kehilangan paket, menyetel codec video untuk meminimalkan kerusakan yang disebabkan oleh kehilangan paket, dan juga menggunakan koreksi kesalahan ke depan. RealNetworks juga menggunakan SureStream, membuat aliran yang berbeda untuk kecepatan koneksi yang berbeda, memprioritaskan key frame, dan membuang beberapa frame dalam proses yang disebut stream thinning. Vosaic (mosaik video) adalah sistem lain yang mengalirkan video melalui aliran yang tidak dapat diandalkan. Server video mentransmisikan video pada frekuensi gambar yang direkam dan klien video memberikan umpan balik mengenai kecepatan penurunan bingkai dan kecepatan penurunan paket. Server kemudian menyesuaikan frekuensi gambar berdasarkan umpan balik klien.

Video modern menyediakan penggunaan streaming adaptif HTTP (HAS) sebagai metode pengiriman video default. Manfaat HTTP dan protokol TCP yang mendasarinya adalah bahwa sistem HAS memiliki transportasi yang andal. Namun, keandalan TCP ada harganya.

Pengiriman ulang paket yang hilang menimbulkan penundaan dan dapat menyebabkan pemblokiran head-of-line.

Pengiriman video HTTP dapat dikirimkan melalui TCP dan QUIC. Bhat dkk. menunjukkan bahwa QUIC tidak memberikan peningkatan kinerja QoE jika dibandingkan dengan TCP. Salah satu faktor mungkin karena pekerjaan tersebut menggunakan implementasi QUIC eksperimental yang disediakan oleh server web Caddy. Langley et al. menunjukkan gambaran kontras, dengan peningkatan QoE yang signifikan dengan QUIC bila dibandingkan dengan TCP. Namun, tidak satu pun dari kedua penelitian tersebut menyelidiki transportasi yang tidak dapat diandalkan.

Pendekatan lain untuk pengiriman video lossy adalah dengan menggunakan protokol yang andal tetapi drop frame untuk menghemat bandwidth. Yahya dkk. mengusulkan sebuah skema di mana setiap frame dalam segmen disampaikan sebagai objek HTTP/2 independen, dan pemutar video dapat menghapus frame yang kurang penting dalam segmen. Namun, pendekatan tersebut tidak mendukung banyak bitrate sehingga tidak dapat bekerja secara efektif dengan variasi jaringan yang tinggi.

Kesimpulan

Pekerjaan sebelumnya melibatkan penyetelan protokol transport atau pengoptimalan lapisan aplikasi, yaitu algoritme ABR. Penulis mengusulkan VOXEL, sistem streaming video baru dengan pengoptimalan lintas lapisan. Hasil empiris menunjukkan bahwa VOXEL mengungguli streaming video canggih dengan mengeksploitasi sinergi lintas lapisan. Desain VOXEL menggunakan wawasan bahwa video dapat mentolerir beberapa kehilangan dan tidak semua bingkai memiliki dampak yang sama pada QoE saat dijatuhkan. Penulis merancang ABR*, algoritme ABR dalam sistem VOXEL yang menggunakan keandalan transpor selektif untuk menghindari buffering ulang tanpa dampak signifikan pada QoE.

BAB 10
SIMULASI VIDEO 360° ADAPTIF

Baru-baru ini konten video 360° semakin populer. Algoritme ABR yang dirancang untuk video biasa tidak selalu berfungsi dengan baik untuk video 360°. Penonton hanya melihat sebagian kecil dari keseluruhan video, yaitu viewport, pada satu waktu. Dengan demikian, menayangkan seluruh video 360° dengan kecepatan bit tinggi dapat menjadi tidak efisien. Ketidakefisienan ini umumnya dikurangi dengan mempartisi area tampilan menjadi petak dan hanya mengirimkan sebagian petak yang ada di area pandang pemirsa. Namun, pendekatan ini dapat menimbulkan masalah jika tidak diterapkan dengan benar. Penonton mengharapkan pengalaman menonton yang mulus saat mengubah pose mereka. Namun, perubahan pose mungkin mengubah subset petak yang terlihat di viewport. Jika konten video untuk petak di viewport baru tidak ada di buffer pemutar, konten tidak akan tersedia untuk pemirsa dan pemirsa mungkin melihat layar kosong. Tantangan ini diperparah oleh fakta bahwa penundaan dari saat penonton menggerakkan kepala hingga saat tampilan bereaksi dapat menyebabkan mabuk perjalanan.

Untuk menayangkan video 360° secara efisien dengan QoE tinggi, pemutar video 360° menggunakan skema prediksi tampilan untuk memprediksi petak mana yang kemungkinan besar berada dalam viewport. Pemain kemudian mengalokasikan sumber daya bandwidth berdasarkan prediksi, menggunakan algoritme ABR 360° untuk mengunduh petak yang berbeda dengan kecepatan bit yang sesuai. Perhatikan bahwa meskipun algoritme ABR 2-D tradisional hanya perlu mengatasi ketidakpastian dalam kondisi jaringan, algoritme ABR 360° perlu mengatasi ketidakpastian baik dalam kondisi jaringan maupun di viewport.

Pengujian merupakan langkah penting dalam pengembangan algoritma adaptasi video 360°. Menguji algoritme 360° pada sistem langsung bahkan lebih sulit daripada menguji video 2-D. Penonton perlu menggunakan head mounted display (HMD), dan performa algoritme sangat bergantung pada perilaku penonton selama setiap sesi. Ini membuat perbandingan yang adil menjadi sulit.

Penulis mengusulkan dan mengembangkan Sabre360, sebuah testbed simulasi untuk menguji algoritme 360° menggunakan pelacakan gerakan kepala yang direkam sebelumnya bersama dengan pelacakan bandwidth yang direkam sebelumnya. Sabre360 memberikan perbandingan yang adil antara algoritme 360° dengan menggunakan pelacakan yang sama untuk setiap sesi pengujian. Penulis juga mengembangkan dua algoritme ABR 360° baru, algoritme berbasis throughput dasar dan algoritme berbasis BOLA.

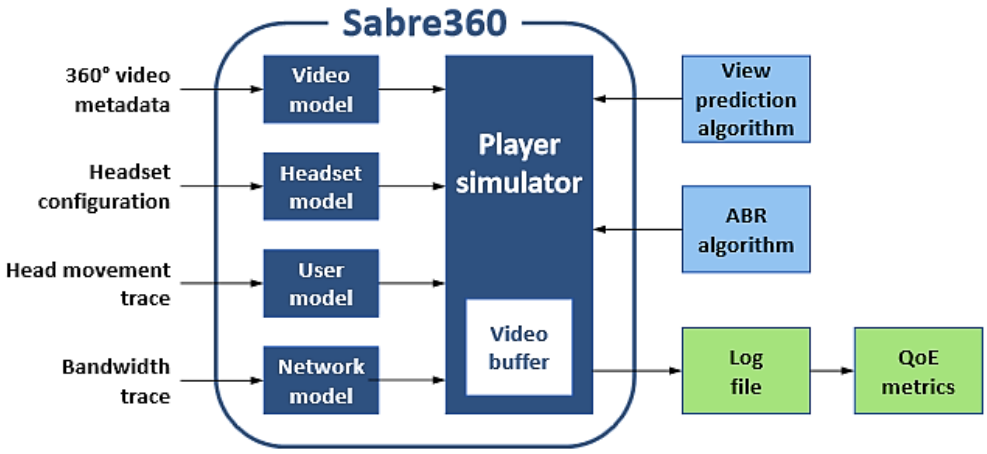
10.1 SABRE360: TESTBED SIMULASI UNTUK VIDEO 360°

Dalam buku ini memperkenalkan Sabre, lingkungan simulasi berbasis Python untuk algoritma ABR 2-D, di Bab 3. Namun, Sabre tidak dapat digunakan untuk mensimulasikan aliran video 360° yang memiliki banyak petak dan viewport yang berbeda. Dalam buku ini

menjelaskan Sabre360, perluasan lingkungan simulasi untuk streaming video 360°. Penulis menjadikan Sabre360 open source dan tersedia untuk umum bagi komunitas di GitHub.

Sabre360 memfasilitasi desain algoritme ABR 360° yang baru. Gambar 10.1 menunjukkan arsitektur sistem. Modul algoritme prediksi tampilan dan modul algoritme bitrate adaptif dapat diuji dengan serangkaian video. Setiap video membutuhkan satu atau lebih jejak pergerakan kepala untuk simulasi. Setiap kombinasi jejak gerakan video dan kepala dapat disimulasikan menggunakan satu set jejak bandwidth. Satu sesi simulasi melibatkan simulasi satu video dengan satu jejak pergerakan head dan satu jejak bandwidth. Sesi simulasi menghasilkan file log peristiwa mendetail yang dapat digunakan untuk menghitung metrik QoE.

Proses simulasi dapat direproduksi dan mengulangi sesi simulasi dengan video yang sama, pelacakan pergerakan head, dan pelacakan bandwidth menghasilkan hasil yang identik.



Gambar 10.1. Arsitektur Sabre360. Sistem yang berbeda dimodelkan secara independen dan pemain inti mengatur sesi video. Algoritma prediksi tampilan dan algoritma bitrate adaptif dapat disediakan oleh pengguna.

Sifat berulang Sabre360 memungkinkan perbandingan yang adil saat mensimulasikan beberapa algoritme ABR. Simulasi sesi video melibatkan banyak komponen yang bekerja bersama-sama yang diatur oleh komponen pusat, simulator pemain. Penulis akan memanggil komponen simulator pemain hanya sebagai pemain di sisa bab ini.

Model Video

Model video merangkum informasi yang berkaitan dengan file video 360°. Meskipun Sabre360 tidak memerlukan file media video 360° yang disandikan untuk mensimulasikan sesi streaming, Sabre360 memerlukan metadata yang menjelaskan ukuran file video dan media. Ini sesuai dengan informasi yang disajikan dalam manifes DASH. Perhatikan bahwa manifes video 360° menggunakan ekstensi deskripsi hubungan spasial (SRD) ke DASH. Model video membaca informasi video dari file JSON yang berisi hal-hal berikut:

- Durasi segmen: durasi segmen dalam ms.
- Konfigurasi ubin: jumlah ubin horizontal dan vertikal dalam video.

- Bitrate: rata-rata bitrate video dalam kbit/s untuk setiap level bitrate. Laju bit rata-rata untuk tingkat kualitas adalah ukuran total bit semua ubin untuk semua segmen pada tingkat kualitas dibagi dengan durasi video.
- Ukuran segmen: ukuran dalam bit pada setiap tingkat kecepatan bit untuk setiap petak di setiap segmen.

Model Jaringan

Model jaringan menyediakan mekanisme untuk mensimulasikan kondisi jaringan berdasarkan jejak jaringan input yang diperoleh dari file JSON. Jejak terdiri dari sejumlah periode, dengan setiap periode memiliki bandwidth tetap dan latensi bolak-balik. Setelah seluruh jejak dikonsumsi, model jaringan mengulanginya dari atas.

Model jaringan mensimulasikan unduhan segmen berurutan melalui jaringan yang analog dengan sesi HTTP. Pemutar meminta unduhan untuk ukuran yang sesuai dengan petak yang diperlukan, dan model jaringan mengembalikan waktu yang dibutuhkan untuk mengunduh. Pemutar juga dapat menunjukkan penundaan, dalam hal ini model jaringan mencari jejak bandwidth lebih dulu.

Ada beberapa kasus algoritme ABR mungkin ingin meninggalkan pengunduhan segmen, misalnya jika kondisi jaringan tiba-tiba memburuk setelah meminta segmen bitrate tinggi. Meninggalkan segmen untuk meminta segmen bitrate rendah adalah mekanisme yang berguna untuk menghindari buffering ulang. Untuk mengaktifkan pengabaian segmen, model jaringan mengirimkan pembaruan progres reguler ke pemain, dan pemain dapat mengabaikan segmen tersebut pada saat itu. Pembaruan progres dikirim saat setidaknya 50 md telah berlalu sejak pembaruan progres terakhir (atau sejak permintaan awal) dan setidaknya 1500 byte telah diunduh sejak pembaruan progres terakhir (atau sejak permintaan awal). Perilaku ini mirip dengan pemain praktis. Misalnya, dash.js menggunakan peristiwa onprogress XMLHttpRequest untuk memicu pemeriksaan apakah akan mengabaikan suatu segmen.

Pemutar video tradisional seperti dash.js memproses segmen secara berurutan, hanya meminta segmen setelah segmen sebelumnya diunduh sepenuhnya. Hal ini menyebabkan kurang dimanfaatkannya bandwidth jaringan, khususnya waktu perjalanan bolak-balik untuk setiap permintaan. Saat streaming video ubin 360°, penggunaan yang kurang ini jauh lebih signifikan. Misalnya, jika durasi segmen adalah 1 detik dan pemain perlu mengunduh sepuluh petak dengan penundaan bolak-balik 50 md, maka pemain kehilangan 500 md atau 50% dari waktu pengunduhan.

Sabre360 memungkinkan dua solusi. Pertama, permintaan untuk beberapa ubin dapat dikelompokkan dan dikirim bersama. Salah satu cara untuk mengimplementasikan ini dalam praktiknya adalah dengan menggunakan server HTTP/2 khusus yang mendorong sejumlah petak sebagai respons terhadap satu permintaan GET yang dibuat dengan sesuai [40]. Kedua, model jaringan memungkinkan perpipaan permintaan, di mana permintaan dapat dikirim sementara permintaan sebelumnya belum sepenuhnya diunduh. Perilaku ini didukung oleh server HTTP/2 biasa.

Model Headset

Video 2-D tradisional dirender pada berbagai perangkat yang memiliki karakteristik berbeda seperti ukuran layar, resolusi, dan rasio aspek. Tampilan yang dipasang di kepala

(HMD) yang digunakan untuk merender video 360° bahkan lebih beragam. Pengubinan video 360° semakin memperumit simulasi streaming video. Mensimulasikan pengiriman video melalui jaringan tidak cukup, dan Sabre360 juga perlu mensimulasikan presentasi video ke pemirsa. Meskipun Sabre360 tidak mengeksplorasi decoding dan rendering video, Sabre360 melacak petak mana yang terlihat oleh pemirsa dan apakah petak terlihat sepenuhnya atau terlihat sebagian.

Model headset mengasumsikan bahwa headset mendukung skema pemasangan yang dijelaskan untuk model video. Namun, area pandang tidak menampilkan semua petak secara bersamaan. Sebagai gantinya, jumlah petak yang terlihat ditentukan menggunakan bidang pandang (FoV) yang dapat dikonfigurasi.

Model headset Sabre360 mendukung proyeksi equirectangular (ERP), proyeksi video 360° yang paling umum. ERP memproyeksikan bola 3-D ke dalam persegi panjang dengan peregangan signifikan di kutub utara dan selatan. Meskipun Sabre360 mendukung berbagai proyeksi melalui model video dan headset modularnya, penulis hanya mempertimbangkan ERP.

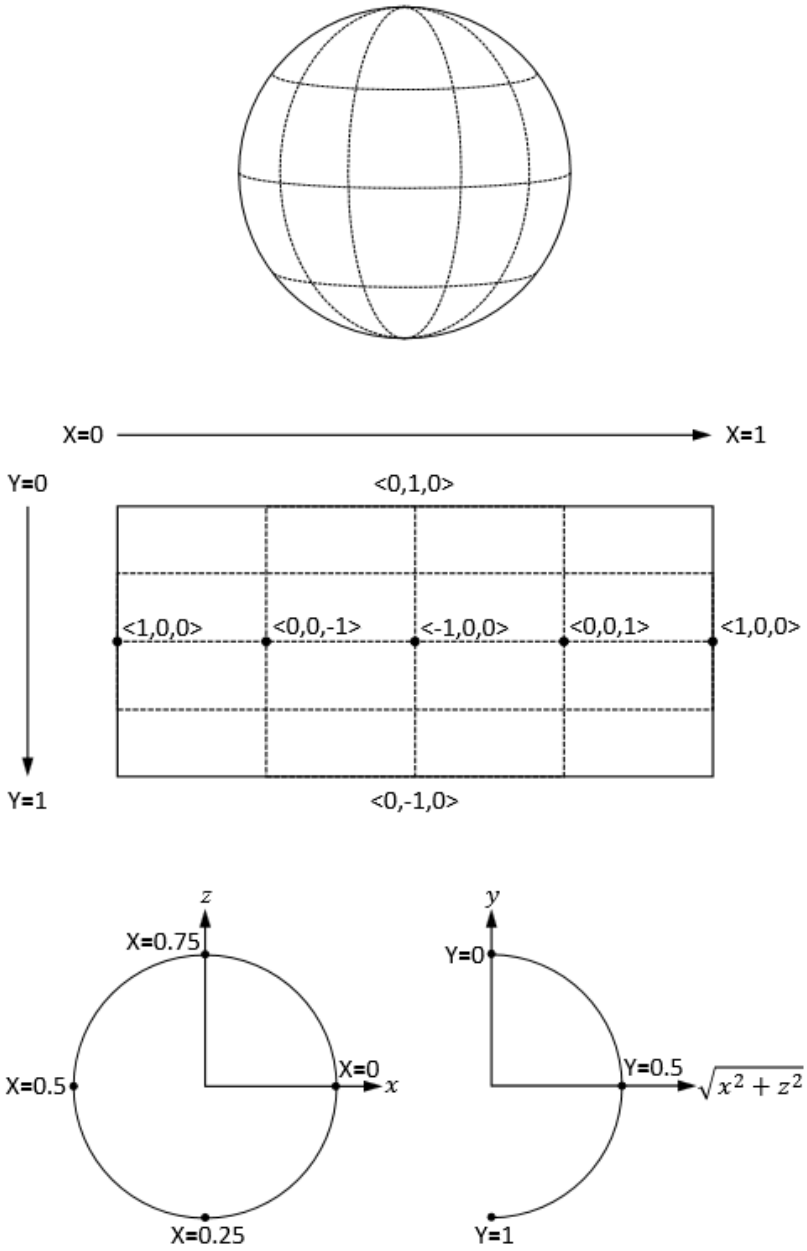
Penulis menggunakan angka empat unit untuk mewakili arah headset. Angka empat mewakili orientasi dan rotasi dalam ruang 3-D. Vektor satuan $\langle x, y, z \rangle$ dalam arah bilangan bulat satuan (qx, qy, qz, qw) dapat dihitung sebagai berikut:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \times qx \times qz + 2 \times qy \times qw \\ 2 \times qy \times qz - 2 \times qx \times qw \\ 1 - 2 \times qx^2 - 2 \times qy^2 \end{bmatrix}$$

Vektor satuan dapat diproyeksikan ke sebuah bola.

Dalam buku ini memproyeksikan video persegi panjang ke bola. Vektor ⟨0, 1, 0⟩ (masing-masing, ⟨0, −1, 0⟩) mewakili pemirsa yang melihat langsung ke utara (masing-masing, selatan). Tanpa kemiringan vertikal, vektor ⟨0, 0, 1⟩ mewakili pemirsa yang melihat ke depan dan vektor ⟨1, 0, 0⟩ mewakili pemirsa yang melihat langsung ke kanan. Gambar 10.2 menunjukkan bagaimana kita memproyeksikan video persegi panjang di bagian dalam sebuah bola. Penulis mewakili setiap titik pada persegi panjang dengan (X, Y) di mana X = 0 (masing-masing, 1) mewakili tepi kiri (masing-masing, kanan), Y = 0 (masing-masing, 1) mewakili tepi atas (masing-masing, bawah). Titik (0, 0,5) diproyeksikan ke vektor ⟨1, 0, 0⟩ dan titik (0,25, 0,5) diproyeksikan ke vektor ⟨0, 0, −1⟩.

Untuk menentukan petak yang terlihat di viewport dari angka empat, pertama-tama kita gunakan (5.1) untuk menentukan vektor satuan. Penulis kemudian menghitung titik (X, Y) dari vektor ⟨x, y, z⟩ sebagai berikut:



Gambar 10.2. Koordinat proyeksi persegi panjang. Bagian dalam bola diproyeksikan ke persegi panjang dengan koordinat (X, Y). Vektor satuan x, y, z dipetakan ke koordinat (X, Y) seperti yang ditunjukkan.

$$\mathrm{X} = \frac{\pi - a\tan 2(-z, -x)}{2\pi}$$

$$\mathrm{Y} = \frac{acos(y)}{\pi}$$

menghitung tepi kiri, kanan, atas dan bawah menggunakan sudut FoV:

$$X_l = X - F_o V_x/(4\pi) \qquad 9.1$$

$$X_r = X + F_o V_x/(4\pi) \qquad 9.2$$

$$Y_t = Y - F_o V_Y/(2\pi) \qquad 9.3$$

$$Y_b = Y + F_o V_Y/(2\pi) \qquad 9.4$$

Pada arah horizontal (X), model headset mampu membungkus tepi yang melewati batas video di (9.1)–(9.2) karena tepi kiri dan kanan terhubung secara melingkar. Pada arah vertikal (Y), model headset memotong tepi yang melewati batas video di (9.3)–(9.4) karena utara dan selatan tidak terhubung. Model headset akhirnya menentukan ubin mana yang termasuk dalam tepi yang dihitung, dan bagian mana dari ubin yang terlihat.

Perhatikan bahwa persegi panjang $(X_l, Y_t) - (X_r, Y_b)$ bukanlah batas yang akurat dari area yang terlihat karena distorsi yang ditimbulkan oleh proyeksi ERP. Namun, arsitektur Sabre360 memungkinkan penghitungan batas yang akurat melalui model headset yang diperbarui.

Tugas lain untuk model headset adalah menentukan cara menangani pemadaman buffer video. Jika petak hilang saat segmen mulai dirender, model headset akan terus diputar tanpa gangguan. Hal ini sesuai dengan penampil yang melihat ubin dibekukan sementara ubin lainnya di area pandang melanjutkan pemutaran.

Model Pengguna

Sementara video 2-D tradisional dapat dirender seluruhnya tanpa input pengguna, rendering video 360° bergantung pada pose kepala penonton. Model pengguna Sabre360 membaca jejak headset dari file JSON dan memperbarui pemutar dengan setiap perubahan pose. Sementara model pengguna yang disertakan dengan Sabre360 mengkodekan pose headset menggunakan angka empat, memperbarui model pengguna untuk menggunakan representasi lain sangatlah mudah. Perhatikan bahwa jejak headset apa pun hanya berguna bila digunakan dengan video yang sama ditampilkan saat direkam.

Simulator Pemain

Simulator pemain menyatukan semua komponen Sabre360. Sesi Sabre360 memerlukan konfigurasi dengan entri berikut:

Ukuran buffer: kapasitas buffer dalam hitungan detik. Pemain tidak akan mengunduh ubin apa pun yang lebih jauh ke masa depan daripada kapasitas buffer dari play head.

Manifes video: file JSON untuk model video.

Pelacakan bandwidth: file JSON untuk model jaringan.

Pelacakan headset: file JSON untuk model pengguna.

Lihat algoritma prediksi: modul Python dengan algoritma prediksi tampilan.

Lihat konfigurasi algoritme prediksi: konfigurasi apa pun yang diperlukan oleh algoritme prediksi tampilan yang dipilih.

Algoritma ABR: modul Python dengan algoritma ABR.

Konfigurasi algoritme ABR: konfigurasi apa pun yang diperlukan oleh algoritme ABR yang dipilih.

Jalur file log: jalur tempat menyimpan file log keluaran.

Selama fase inisialisasi, pemutar menginisialisasi model video dengan JSON manifes, model jaringan dengan JSON jejak bandwidth, dan model pengguna dengan JSON jejak headset. Pemain juga menginisialisasi buffer kosong untuk setiap petak. Kemudian pemain mengulangi loop berikut hingga seluruh video diputar:

1. *Perkirakan throughput menggunakan throughput estimator*. Perhatikan bahwa hanya model jaringan yang diizinkan untuk mengakses jejak bandwidth, semua komponen lainnya hanya dapat menggunakan estimasi throughput.
2. *Meminta algoritme ABR untuk suatu tindakan*. Algoritme ABR dapat menanyakan algoritme prediksi tampilan untuk memilih tindakan. Tindakannya akan menjadi indeks segmen, indeks petak, dan tingkat bitrate. Algoritme ABR diizinkan untuk mengunduh ubin baru atau memutakhirkan ubin yang diunduh sebelumnya dengan kecepatan bit yang lebih tinggi.
3. *Panggil model jaringan dengan ukuran unduhan.* Model jaringan menggunakan panggilan balik untuk mengirim pembaruan progres setiap 50 md hingga seluruh unduhan selesai.
4. *Pada setiap pembaruan progres dan penyelesaian unduhan*, periksa model pengguna untuk perubahan pose. Beri tahu algoritme prediksi tampilan dengan setiap perubahan pose. Pertahankan level play head dan buffer pada level yang benar selama perubahan pose.
5. *Pada setiap pembaruan progres*, setelah menyelesaikan Langkah 4, lakukan kueri pada algoritme ABR untuk menentukan apakah akan membatalkan pengunduhan. Jika algoritme ABR memutuskan untuk mengabaikan, maka perintahkan model jaringan untuk membatalkan pengunduhan dan ulangi dari Langkah 1.
6. *Ketika pengunduhan berhasil diselesaikan*, setelah menyelesaikan Langkah 4, perbarui status buffer. Kemudian ulangi dari Langkah 1.

Langkah-langkah di atas disederhanakan dan menghilangkan perpipaan. Untuk menangani perpipaan, pemutar menggunakan penaksir throughput untuk memperkirakan kapan unduhan akan selesai. Dari perkiraan waktu penyelesaian pengunduhan, penulis kurangi 50 ms atau dua kali perkiraan waktu pulang pergi, mana saja yang lebih besar. Saat ini, pemain mengkueri algoritme ABR menggunakan status sistem yang diprediksi, dan mengirimkan kueri ke model jaringan terlebih dahulu. Pemain dapat memiliki beberapa permintaan yang disalurkan dengan cara ini

10.2 ALGORITMA ADAPTIF

Sabre360 membantu pengembangan cepat algoritme adaptif 360°. Gambar 10.1 menunjukkan bahwa Sabre360 memerlukan dua algoritme input, algoritme prediksi tampilan dan algoritme ABR. Sabre360 juga menyertakan contoh implementasi untuk keduanya. Dengan demikian, pengguna dapat mensimulasikan sesi video menggunakan Sabre360 di luar kotak. Pengguna kemudian dapat mengganti salah satu algoritme atau keduanya.

Lihat algoritma prediksi

Efisiensi sistem streaming 360° bergantung pada prediksi tampilan yang akurat. Sabre360 mengambil implementasi modul Python dari algoritma prediksi tampilan sebagai masukan. Algoritme menerima peristiwa perubahan pose di masa lalu, dan dapat ditanyakan tentang masa depan. Algoritme prediksi tampilan memiliki dua metode untuk berinteraksi dengan Sabre360. Pertama, ia memiliki metode yang memungkinkan Sabre360 memberi tahu algoritme prediksi tampilan setiap kali pose pengguna berubah. Kedua, ia memiliki metode untuk menanyakan prediksi tampilan untuk segmen tertentu di masa mendatang. Algoritme prediksi tampilan menampilkan vektor probabilitas, yang menunjukkan kemungkinan setiap petak berada di area pandang yang terlihat untuk segmen yang diminta. Perhatikan bahwa prediksi segmen di masa mendatang mungkin berubah saat model pengguna melanjutkan pelacakan headset.

Prediksi tampilan berdasarkan grafik navigasi

Sabre360 menyertakan contoh algoritma prediksi tampilan berdasarkan grafik navigasi [39]. Grafik navigasi adalah struktur data yang menyandikan perubahan port tampilan sebelumnya, memungkinkan estimasi efisien area pandang di masa mendatang. Grafik navigasi berfungsi dalam dua mode, lintas pengguna dan pengguna tunggal.

Dalam mode cross-user, setiap node (disebut view) dalam grafik adalah tuple yang terdiri dari indeks segmen dan kumpulan semua petak yang terlihat di titik mana pun selama seluruh durasi segmen. Sisi dari u ke v diberi bobot oleh probabilitas bahwa pemirsa dalam tampilan u pergi ke tampilan v. Perhatikan bahwa indeks segmen dalam v adalah satu ditambah indeks segmen dalam u. Grafik navigasi lintas pengguna diisi dengan menggabungkan data dari beberapa penonton yang menonton video tertentu. Selama sesi streaming video, grafik navigasi dapat digunakan dengan tampilan saat ini untuk memberikan daftar potensi tampilan berikutnya, beserta probabilitasnya. Proses tersebut dapat diulangi untuk memprediksi tampilan beberapa segmen ke depan. Probabilitas bahwa petak tertentu akan terlihat dapat dihitung dengan menjumlahkan probabilitas semua tampilan yang menyertakan petak tersebut. Perhatikan bahwa grafik navigasi bisa jarang dan tampilan saat ini tidak dijamin berada di grafik navigasi. Dalam kasus seperti itu, tampilan dengan jumlah petak terbanyak yang berpotongan dengan tampilan saat ini akan digunakan.

Dalam mode pengguna tunggal, setiap simpul (tampilan) dalam grafik adalah kumpulan semua petak yang terlihat di titik mana pun selama seluruh durasi segmen. Setiap sesi streaming dimulai dengan grafik kosong, dan grafik diperbarui setelah setiap segmen. Perhatikan bahwa tampilan tidak menyertakan indeks segmen. Dengan demikian, grafik navigasi satu pengguna menangkap pola gerakan kepala yang terpisah dari konten.

Grafik navigasi lintas pengguna digunakan bersama dengan grafik navigasi pengguna tunggal. Selama segmen pertama, grafik single-user masih kosong sehingga hanya grafik cross-user yang digunakan untuk prediksi. Setelah setiap segmen diputar, algoritme memeriksa ketepatan prediksi menggunakan kedua grafik secara terpisah. Grafik yang menghasilkan presisi lebih tinggi kemudian digunakan untuk prediksi segmen berikutnya. Jika pola tontonan penonton cocok dengan tren umum, grafik lintas pengguna cenderung akan mengambil alih.

Namun, jika pemirsa tidak mengikuti tren umum, grafik pengguna tunggal kemungkinan besar akan mengambil alih.

Implementasi Sabre360 dari prediksi tampilan berbasis grafik navigasi dapat memasukkan grafik navigasi lintas pengguna dari file JSON yang diteruskan yang ditunjukkan oleh konfigurasi algoritme prediksi tampilan.

Algoritma ABR

Algoritme ABR mengontrol semua unduhan video. Sabre360 mengambil implementasi modul Python dari algoritma ABR sebagai masukan. Sabre360 menanyakan algoritme ABR saat siap untuk unduhan berikutnya. Algoritme ABR dapat memperoleh informasi berikut dari pemain:

Waktu: waktu dinding dan waktu main head.

Konten buffer: kecepatan bit untuk setiap petak dalam buffer pemutaran.

Pose: pose headset saat ini yang diperoleh dari model pengguna.

Lihat prediksi: algoritme ABR dapat menanyakan algoritme prediksi tampilan melalui pemutar.

```
1: function GetAction
2:     tp ← throughput estimate
3:     b ← tp × segment duration
4:     s ← first segment after play head with at least one empty tile
5:     b ← b − (bits previously used to download any tiles in s)
6:     p_s ← ViewportPredictor.GetTileProbabilities(s)
7:     q ← AllocateQuality(s, b, p_s)
8:     t ← arg max_t p_s[t] subject to tile t in s being empty
9:     return (s, t, q[t])
10: end function
```

Gambar 10.3. Algoritme ABR 360° dasar.

Algoritme ABR mengembalikan tindakan, yang menunjukkan segmen, petak, dan kecepatan bit untuk diunduh. Selama pengunduhan, pemain memperoleh peristiwa kemajuan dari model jaringan dan menanyakan algoritme ABR apakah akan melanjutkan pengunduhan atau mengabaikan. Pemain akhirnya memberi tahu algoritme ABR tentang unduhan yang berhasil, yang menunjukkan ukuran unduhan, waktu unduhan, dan waktu ke byte pertama.

Algoritme ABR dapat melakukan estimasi throughput berdasarkan laporan unduhan yang berhasil diterimanya dari pemutar. Sabre360 juga menyertakan estimasi throughput primitif menggunakan rata-rata pergerakan berbobot eksponensial. Algoritme ABR dapat menggunakan Sabre360 primitif atau melakukan estimasi throughputnya sendiri.

Perhatikan bahwa algoritme ABR tidak memiliki akses ke pelacakan jaringan yang digunakan oleh model jaringan karena algoritme yang diterapkan hanya dapat memperkirakan throughput dari riwayat unduhan.

Sabre360 menyertakan dua contoh algoritme ABR, algoritme berbasis throughput baseline dan algoritme berbasis BOLA. Pengguna dapat menggunakan salah satu contoh algoritma atau mengimplementasikannya sendiri.

Algoritma ABR dasar

Saat streaming video 2-D tradisional, algoritme THROUGHPUT dapat digunakan sebagai algoritme dasar karena sederhana namun berkinerja cukup baik. Penulis sekarang merancang algoritme berbasis throughput dasar untuk video 360°.

1: **function** ALLOCATEQUALITY$(s, b, \mathbf{p}_s)$
2: $\quad \mathbf{z}_m \leftarrow$ segment sizes for s ▷ for $1 \le m \le M$
3: $\quad \mathbf{q} \leftarrow \langle 1, 1, \ldots, 1 \rangle$ ▷ start with lowest quality $m = 1$
4: $\quad \mathcal{T} \leftarrow$ set of empty tiles in s
5: $\quad b \leftarrow b - \sum_{t \in \mathcal{T}} \mathbf{z}_1[t]$
6: $\quad t \leftarrow \arg\min_t(\mathbf{z}_{\mathbf{q}[t]+1}/\mathbf{p}_s[t])$ subject to $(\mathbf{q}[t] + 1 \le M)$, $(\mathbf{z}_{\mathbf{q}[t]+1}[t] - \mathbf{z}_{\mathbf{q}[t]}[t] \le b)$
7: $\quad$ **if** we found some t in line 6 **then**
8: $\quad\quad b \leftarrow b - (\mathbf{z}_{\mathbf{q}[t]+1}[t] - \mathbf{z}_{\mathbf{q}[t]}[t])$
9: $\quad\quad \mathbf{q}[t] \leftarrow \mathbf{q}[t] + 1$
10: $\quad\quad$ **repeat** line 6
11: $\quad$ **end if**
12: $\quad$ **return q**
13: **end function**

Gambar 10.4. Penolong algoritme ABR 360° Baseline.

Gambar 10.3–10.4 menjelaskan algoritme. Ide utamanya adalah menggunakan estimasi throughput untuk memperkirakan jumlah bit yang tersedia untuk mengunduh segmen tertentu, lalu menggunakan probabilitas petak dari algoritme prediksi tampilan untuk mengalokasikan lebih banyak bit ke petak yang lebih penting. Algoritme ini dirancang untuk mengunduh satu petak dalam satu waktu tanpa menyimpan informasi status apa pun.

Untuk menentukan tindakan selanjutnya, algoritma Baseline pertama-tama meminta estimator throughput primitif dan mengalikan estimasi throughput dengan durasi segmen untuk mendapatkan anggaran bit segmen. Algoritma kemudian mencari segmen pertama di buffer dengan setidaknya satu petak hilang. Ukuran dalam bit petak apa pun dari segmen s yang sudah ada di buffer kemudian dikurangi dari anggaran bit. Anggaran bit kemudian didistribusikan ke semua petak yang tersisa dari segmen s sebagai berikut. Biarkan b menjadi anggaran bit dan pt menjadi probabilitas bahwa petak t ada di viewport. Maka alokasi bit algoritma Baseline untuk petak t adalah $bp_t / \sum t^1 p^1$ Gambar 5.4 menunjukkan bagaimana algoritma menangani kuantisasi.

Algoritma ABR berbasis BOLA

Sabre360 juga menyertakan algoritme ABR 360° berdasarkan BOLA, yang penulis sebut BOLA-TS. BOLA-TS mirip dengan BOLA-E tetapi dengan perbedaan sebagai berikut:

- **Lihat prediksi:** BOLA-TS menanyakan algoritme prediksi tampilan di awal setiap panggilan.

- **Satu penyangga per petak**: BOLA-TS menghitung tingkat penyangga untuk setiap petak secara terpisah. Hal ini memungkinkan BOLA-TS memberikan prioritas lebih tinggi pada petak yang lebih penting. Misalnya, ia mungkin mengunduh petak dengan arah maju untuk lima segmen dalam buffer sementara hanya mengunduh petak dengan arah berlawanan untuk dua segmen dalam buffer.
- **Pembobotan fungsi tujuan:** BOLA-TS menghitung fungsi tujuan ρ untuk setiap pengunduhan petak potensial. Kemudian setiap nilai dikalikan dengan probabilitas ubin tertentu akan dilihat.
- **Penggantian petak:** Pertimbangkan petak yang diunduh dengan kecepatan bit rendah karena probabilitas penayangannya yang rendah. Namun, penonton mungkin mengubah pose mereka dan secara signifikan meningkatkan kemungkinan melihat petak. Dalam kasus seperti itu, BOLA-TS memiliki opsi untuk memutakhirkan ubin di buffer. Perhatikan bahwa algoritma FAST-SWITCHING tidak bekerja secara langsung karena penulis hanya ingin mengganti petak berdasarkan informasi prediksi tampilan terbaru.
- **Baseline keamanan bitrate rendah:** Aturan buffer yang tidak memadai dirancang untuk video 2-D. Penulis memperluas gagasan untuk mengunduh garis dasar keamanan bitrate rendah untuk video 360° agar tidak merender petak kosong. Dengan algoritme keamanan baru, pertama-tama penulis menentukan segmen paling awal dengan setidaknya satu petak yang hilang. Penulis juga menghitung jumlah bit yang diperlukan untuk mengunduh ubin yang hilang dengan kecepatan bit terendah. Penulis kemudian menggunakan estimasi throughput untuk menentukan berapa banyak bit yang diharapkan dapat diunduh dengan aman hingga segmen mulai dirender. BOLA-TS tidak diizinkan untuk memulai pengunduhan yang mungkin mencegah ubin diunduh dengan aman.

Sementara sebagian besar perubahan di atas relatif sederhana untuk diterapkan, penulis memerlukan beberapa analisis untuk menyesuaikan penggantian ubin dalam kerangka kerja utilitas BOLA. Penulis ingin menentukan utilitas yang adil untuk membandingkan unduhan pengganti dengan unduhan baru.

Pertimbangkan dua tingkat kualitas m_1 dan m_2 untuk segmen tertentu yang memiliki utilitas v_1 dan v_2 dan ukuran masing-masing bit S_1dan S_2, di mana $v_1 < v_2 \, dan \, S_1 < S_2$. Lalu ada beberapa level buffer Q′ sehingga ρ_{a1}=1 = ρ_{a2}=1. Jadi, di QJ, BOLA tidak memiliki preferensi di antara keduanya. Wawasan utama adalah bahwa jika penulis dapat memutakhirkan segmen dari kualitas m_1 ke kualitas m_2 dengan mengunduh bit ($S_1 - S_2$), maka penulis berharap BOLA tidak memiliki preferensi antara pemutakhiran dan salah satu dari dua segmen pada tingkat buffer Q', yaitu :

$$\rho_{a_1 - 1} = \rho_{a_2 - 1} = \rho_{1 \to 2}$$

Kita dapat menggunakan wawasan ini untuk menghitung nilai utilitas wajar untuk operasi penggantian. Dimulai dengan m_1 dan m_2, diperoleh:

$$\frac{V(v_1+\gamma p)-Q'}{S_1}=\frac{V(v_2+\gamma p)-Q'}{S_2}$$

Memecahkan, kita mendapatkan:

$$Q'=V(v_1+\gamma p)=\frac{VS_1(v_2+v_1)}{S_2-S_1}$$

Asumsikan untuk saat ini bahwa ada opsi pemutakhiran yang efisien. Artinya, kita dapat memutakhirkan segmen dari level kualitas m_1 ke level m_2 dengan mengunduh bit (S_1-S_2),yang tepat. Biarkan pemutakhiran memiliki beberapa utilitas $v_1 \to 2$. Penulis sekarang dapat menulis ekspresi untuk fungsi tujuan untuk pemutakhiran sebagai:

$$\rho_{1\to 2}=\frac{Vv_{1\to 2}-Q}{S_2-S_1}$$

Perhatikan bahwa kita tidak memasukkan suku γp karena $v_1 \to 2$ masih belum diketahui dan berpotensi memasukkan suku tersebut.

Kita tahu persamaan diatas bahwa $P_{a2=1}=P^{1\to 2}$

$$\frac{V(v_1+\gamma p)-Q'}{S_2}=\frac{Vv_{1\to 2}-Q'}{S_2-S_1}$$

Maka kita akan mendapatkan:

$$v_{1\to 2}=v_2-\frac{S_1(v_2-v_1)}{S_2-S_1}+\gamma p$$

Dan

$$\rho_{1\to 2}=\frac{V\left(v_2-\frac{(v_2-v_1)S_1}{S_2-S_1}+\gamma p\right)-Q}{S_2-S_1}$$

Perhatikan pernyataan diatas menyertakan suku γp.

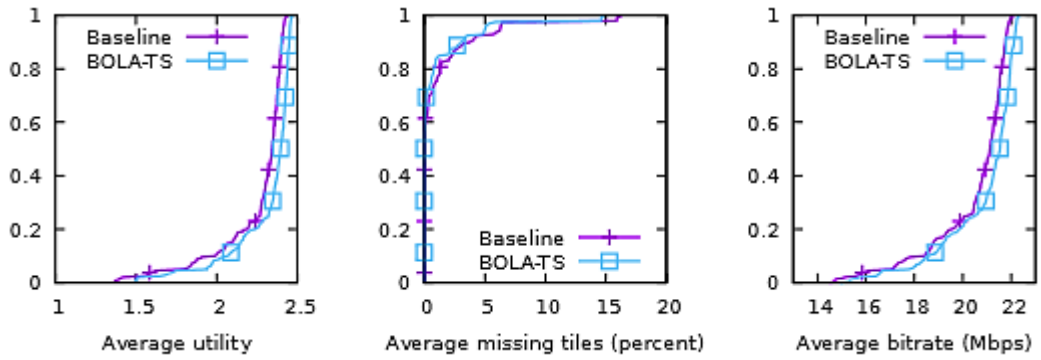
Namun, peningkatan dari level kualitas m_1 ke level m_2 hanya memiliki ukuran (S_1-S_2),bit saat menggunakan pengkodean yang dapat diskalakan. Jika tidak, pemutakhiran memiliki ukuran bit S_2. Jika video yang dapat diskalakan tidak digunakan, penulis dapatkan

$$\rho'_{1\to 2}=\frac{V\left(v_2-\frac{(v_2-v_1)S_1}{S_2-S_1}+\gamma p\right)-Q}{S_2}$$

BOLA-TS menggunakan perhitungan untuk menghitung secara wajar utilitas untuk segmen pengganti.

10.3 MENGEVALUASI ALGORITMA ADAPTIF 360° MENGGUNAKAN SABRE360

Penulis sekarang menunjukkan bagaimana Sabre360 dapat digunakan untuk mengevaluasi algoritme adaptif dengan membandingkan algoritme Baseline ABR dengan BOLA-TS menggunakan algoritme prediksi tampilan berbasis grafik navigasi. Penulis menggunakan 48 jejak headset untuk video Google Spotlight-HELP. Penulis menggunakan 42 jejak pertama untuk mengisi grafik navigasi lintas pengguna dan 6 jejak sisanya untuk evaluasi. Video dikodekan pada lima kecepatan bit antara 2 dan 23 Mbps, disusun menjadi ubin 4 × 4 dan disegmentasi dengan segmen 1s. Dalam evaluasi penulis, penulis menggunakan 40 jejak 4G. Sabre360 memberikan file log untuk setiap sesi, dan penulis memproses file log untuk menghasilkan informasi metrik. Selama percobaan penulis, penulis berfokus pada tiga metrik yang dijelaskan di bawah ini.



Gambar 10.5. CDF untuk 6 penonton untuk 40 jejak 4G dengan kapasitas buffer 5 detik. BOLA-TS memiliki utilitas rata-rata 2,31, ubin hilang rata-rata 0,9%, dan bitrate rata-rata 20,9 Mbps. Baseline memiliki utilitas rata-rata 2,25, ubin hilang rata-rata 1,2%, dan bitrate rata-rata 20,5 Mbps.

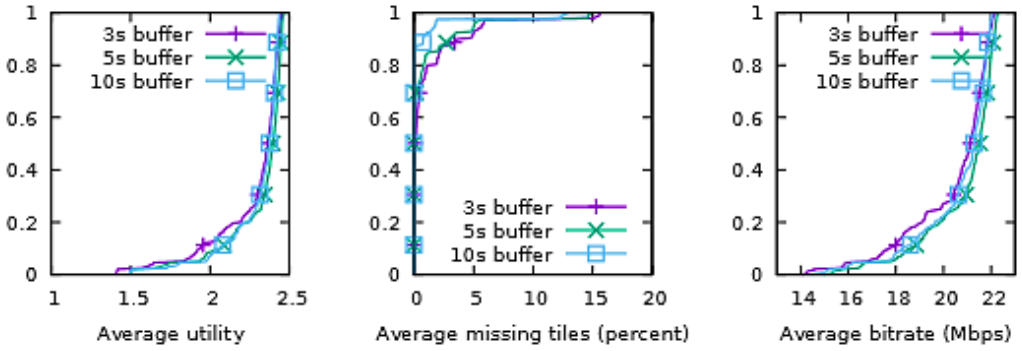
Utilitas rata-rata: Penulis menghitung utilitas BOLA sebagai $\ln(S_m/S_1)$. Model headset tidak melakukan rebuffer saat kehilangan satu petak, tetapi penulis tetap memberikan penalti sebesar $-\upsilon M$ untuk setiap petak yang hilang. Untuk menghitung utilitas rata-rata, pertama-tama kita menghitung utilitas rata-rata untuk setiap waktu yang terlihat setiap saat. Kemudian penulis menghitung rata-rata utilitas seketika di seluruh sesi video. Perhatikan bahwa penulis perhitungan utilitas rata-rata tidak dikondisikan oleh batas segmen.

Tabel 10.1. Metrik QoE untuk algoritme Baseline dan BOLA-TS untuk 6 penonton untuk 40 jejak 4G.

	Baseline			BOLA-TS		
Buffer Capacity (detik)	Utility	Missing tiles (%)	Bitrate (Mbps)	Utility	Missing tiles (%)	Bitrate (Mbps)
3	2.21	1.93	20.3	2.27	1.17	20.4
5	2.25	1.16	20.5	2.31	0.94	20.9
10	2.28	0.64	20.6	2.30	0.44	20.6

Rata-rata petak yang hilang: Saat memutar video, pengguna dapat melihat banyak petak. Petak apa pun di viewport yang tidak memiliki data video dianggap sebagai petak yang hilang. Mirip dengan utilitas, pertama-tama penulis menghitung rata-rata setiap saat, lalu menghitung rata-rata hasilnya di seluruh video.

Laju bit rata-rata: Sekali lagi, pertama-tama penulis menghitung rata-rata di semua petak yang terlihat, dengan petak yang hilang memiliki laju bit 0, lalu menghitung rata-rata hasilnya di seluruh video. Perhatikan bahwa bitrate rata-rata mungkin lebih tinggi daripada bandwidth jaringan. Misalnya, jika pemutar video berhasil melakukan streaming video 20 Mbps hanya dengan mengunduh setengah petak, maka bitrate rata-rata masih 20 Mbps meskipun hanya menggunakan kapasitas jaringan 10 Mbps.



Gambar 10.6. CDF untuk 6 pemirsa untuk 40 jejak 4G menggunakan BOLA-TS.

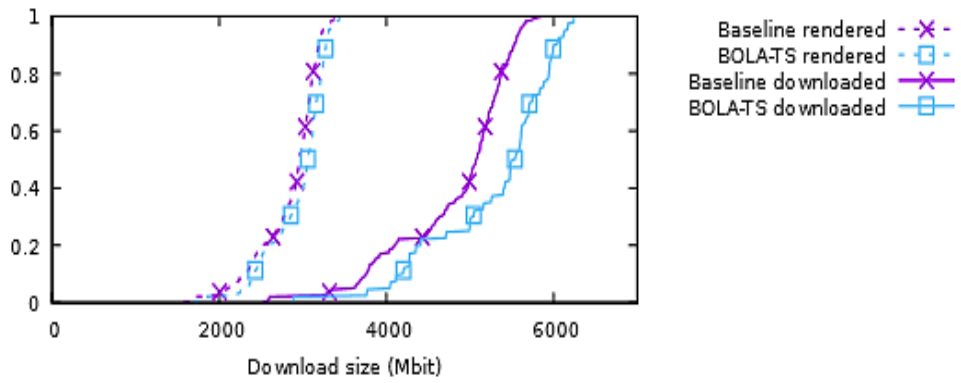
Penulis pertama kali menggunakan Sabre360 untuk mensimulasikan algoritme Baseline dan BOLA-TS dengan kapasitas buffer 5 detik untuk 6 pemirsa dan 40 pelacakan 4G, menguji setiap pelacakan headset pemirsa dengan setiap pelacakan jaringan. Gambar 10.6 menunjukkan utilitas rata-rata, ubin yang hilang, dan kecepatan bit. Nilai rata-rata ditunjukkan pada Tabel 10.1. BOLA-TS memiliki utilitas 3% lebih tinggi daripada Baseline, dengan ubin yang hilang 19% lebih sedikit dan bitrate 2% lebih tinggi.

Penulis mengulangi percobaan dengan kapasitas buffer 3s dan 10s. Gambar 10.6 membandingkan BOLA-TS untuk kapasitas penyangga yang berbeda. Nilai rata-rata ditunjukkan pada Tabel 10.1. Perhatikan bahwa saat meningkatkan buffer dari 3 detik menjadi 5 detik memberikan QoE yang lebih baik di semua metrik, meningkatkan buffer dari 5 detik menjadi 10 detik menyebabkan BOLA-TS kehilangan beberapa bitrate dan utilitas. Dengan

video 2-D tradisional, penulis berharap mendapatkan QoE yang lebih baik untuk buffer yang lebih besar.

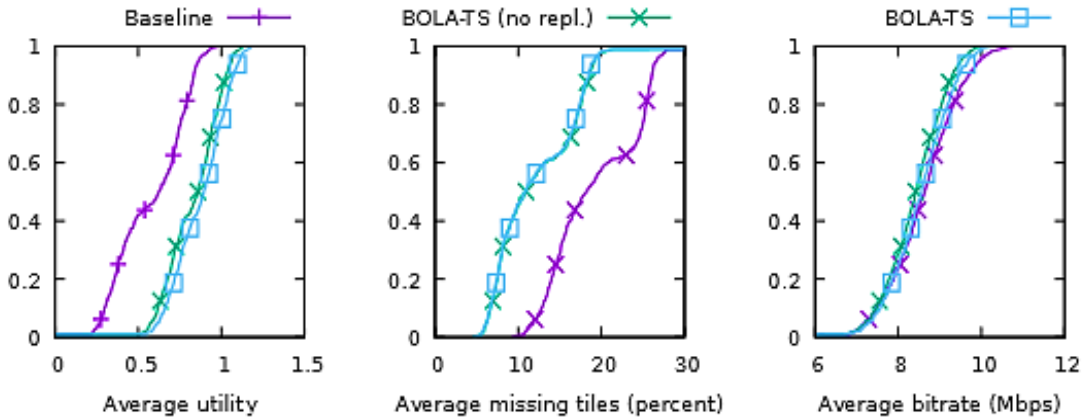
Masuk akal untuk mengharapkan tren serupa untuk video 360°. Namun, prediksi tampilan akan kehilangan akurasi di masa mendatang. Hal ini dapat menyebabkan algoritme ABR menghabiskan sumber daya bandwidth untuk unduhan yang kurang berguna. Dengan buffer yang lebih kecil, BOLA-TS memiliki prediksi tampilan yang lebih akurat dan menggunakan sumber daya secara lebih efektif. Di sisi lain, penyangga yang lebih besar masih mengurangi ubin yang hilang.

Sabre360 juga merekam informasi ukuran unduhan. Pengunduhan yang tidak terpakai hampir tidak dapat dihindari saat streaming video 360°. Karena prediksi tampilan tidak bisa 100% akurat, penulis berharap mengunduh beberapa petak yang tidak pernah direnderer. Gambar 10.7 menunjukkan CDF untuk jumlah total bit yang diunduh selama percobaan sebelumnya dengan kapasitas buffer 5 detik. Penulis juga mengukur jumlah total bit yang digunakan untuk mengunduh petak yang berguna, yaitu petak yang berada di viewport di beberapa titik. Algoritme Baseline menggunakan 59% dari bit yang diunduh sementara BOLA-TS menggunakan 56% dari bit yang diunduh.



Gambar 10.7. CDF ukuran unduhan untuk 6 penonton untuk 40 trek 4G dengan kapasitas buffer 5 detik. Ukuran unduhan adalah jumlah total bit yang diunduh dalam satu sesi. Ukuran yang direnderer adalah subset dari total unduhan termasuk hanya petak yang diunduh yang berada di viewport di beberapa titik. Algoritme Baseline menggunakan 59% dari bit yang diunduh sementara BOLA-TS menggunakan 56% dari bit yang diunduh.

Sabre360 juga dapat digunakan dengan jejak sintetik untuk menguji skenario tertentu. Penulis menguji BOLA-TS dengan dan tanpa penggantian ubin. Karena penulis tidak melihat perbedaan signifikan saat menggunakan jejak 4G, penulis membuat 100 jejak sintetik. Di setiap jejak sintetik, penulis memiliki 5 detik bandwidth rendah (di bawah 2 Mbps) diikuti oleh 10 detik bandwidth tinggi (di atas 20 Mbps), diikuti lagi oleh 5 detik bandwidth rendah dan seterusnya.



Gambar 10.8. CDF untuk 6 penonton untuk 100 jejak sintetik dengan variasi bandwidth tinggi menggunakan kapasitas buffer 3 detik. Penulis menggunakan BOLA-TS dengan penggantian diaktifkan (default) dan dinonaktifkan.

Gambar 10.8 menunjukkan bahwa penggantian meningkatkan bitrate dan utilitas. Menonaktifkan penggantian ubin kehilangan utilitas 4,9%, bitrate 1,4%, dan memiliki ubin yang hilang 0,6% lebih banyak. Meskipun penulis tidak mengharapkan penggantian untuk meningkatkan metrik ubin yang hilang, perbedaannya tidak signifikan. Mengukur ukuran unduhan, BOLA-TS dengan penggantian menggunakan 60,9% dari total bit yang diunduh sementara BOLA-TS tanpa penggantian menggunakan 61,3%. Algoritme Baseline memiliki utilitas 33,8% lebih sedikit, bitrate 0,9% lebih banyak, dan 59% lebih banyak ubin yang hilang jika dibandingkan dengan BOLA-TS. Perhatikan bahwa ubin yang hilang memiliki efek yang lebih tinggi pada QoE daripada bitrate yang lebih tinggi. Penulis mengulangi percobaan dengan kapasitas buffer 5s dan 10s. Tidak ada perbedaan yang signifikan antara BOLA-TS dengan penggantian dan BOLA-TS tanpa penggantian untuk kapasitas penyangga yang lebih besar.

KESIMPULAN

Buku ini merancang dan mengimplementasikan Sabre360, tempat pengujian simulasi untuk algoritme adaptasi video 360°. Dalam buku ini mendemonstrasikan bagaimana Sabre360 dapat menjadi alat yang berguna untuk merancang dan membandingkan algoritme baru. Penulis juga mengembangkan dua algoritme ABR 360° dan membandingkan kinerjanya menggunakan Sabre360.

DAFTAR PUSTAKA

Akhtar, Zahaib, Nam, Yun Seong, Govindan, Ramesh, Rao, Sanjay, Chen, Jessica, Katz-Bassett, Ethan, Ribeiro, Bruno, Zhan, Jibin, and Zhang, Hui. Oboe: Auto-tuning video abr algorithms to network conditions. In Proc. ACM SIG- COMM (2018).

Bentaleb, Abdelhak, Timmerer, Christian, Begen, Ali C, and Zimmermann, Roger. Bandwidth prediction in low-latency chunked streaming. In Proceed- ings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (2019), pp. 7–13.

Bentaleb, Abdelhak, Timmerer, Christian, Begen, Ali C, and Zimmermann, Roger. Performance analysis of acte: A bandwidth prediction method for low- latency chunked streaming. ACM Transactions on Multimedia Computing, Com- munications, and Applications (TOMM) 16, 2s (2020), 1–24.

Bertsekas, Dimitri. Dynamic programming and optimal control, vol. 1. Athena Scientific Belmont, MA, 1995.

Bhat, Divyashri, Rizk, Amr, and Zink, Michael. Not so QUIC: A performance study of DASH over QUIC. In Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video (2017), ACM, pp. 13–18.

Chen, Zhigang, Tan, See-Mong, Campbell, Roy H, and Li, Yongcheng. Real time video and audio in the world wide web. In Fourth International World Wide Web Conference (1995), Citeseer, pp. 333–348.

De Cicco, Luca, Caldaralo, Vito, Palmisano, Vittorio, and Mascolo, Saverio. Elastic: a client-side controller for dynamic adaptive streaming over HTTP (DASH). In Packet Video Workshop (PV), 2013 20th International (2013), IEEE, pp. 1–8.

Dobrian, Florin, Sekar, Vyas, Awan, Asad, Stoica, Ion, Joseph, Dilip, Ganjam, Aditya, Zhan, Jibin, and Zhang, Hui. Understanding the impact of video quality on user engagement. In ACM SIGCOMM Computer Communication Review (2011), vol. 41, ACM, pp. 362–373.

Feamster, Nick, and Balakrishnan, Hari. Packet Loss Recovery for Streaming Video. In 12th International Packet Video Workshop (April 2002).

Handley, Mark, and Perkins, Colin. Rfc2736: Guidelines for writers of rtp pay- load format specifications, 1999.

Huang, Te-Yuan, Johari, Ramesh, McKeown, Nick, Trunnell, Matthew, and Wat- son, Mark. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In Proc. ACM SIGCOMM (2014).

Huang, Te-Yuan, Johari, Ramesh, McKeown, Nick, Trunnell, Matthew, and Wat- son, Mark. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. ACM SIGCOMM Computer Communication Review 44, 4 (2015), 187–198.

Huynh-Thu, Quan, and Ghanbari, Mohammed. Scope of validity of PSNR in image/video quality assessment. Electronics letters 44, 13 (2008), 800–801.

Jiang, Junchen, Sekar, Vyas, and Zhang, Hui. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. IEEE/ACM Transactions on Networking (ToN) 22, 1 (2014), 326–340.

Krishnan, S Shunmuga, and Sitaraman, Ramesh K. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. In Proc. ACM IMC (2012).

Langley, Adam, Riddoch, Alistair, Wilk, Alyssa, Vicente, Antonio, Krasic, Charles, Zhang, Dan, Yang, Fan, Kouranov, Fedor, Swett, Ian, Iyengar, Janard- han, et al. The QUIC transport protocol: Design and internet-scale deployment. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (2017), ACM, pp. 183–196.

Law, Will. Ultra-low-latency streaming using chunked-encoded and chunked- transferred cmaf. Tech. rep., Technical report, Akamai, 2018.

Li, Zhi, Zhu, Xiaoqing, Gahm, Joshua, Pan, Rong, Hu, Hao, Begen, Ali, and Oran, David. Probe and adapt: Rate adaptation for HTTP video streaming at scale. IEEE JSAC 32, 4 (2014), 719–733.

Maggs, Bruce M, and Sitaraman, Ramesh K. Algorithmic nuggets in content delivery. ACM SIGCOMM Computer Communication Review 45, 3 (2015), 52– 66.

Mao, Hongzi, Netravali, Ravi, and Alizadeh, Mohammad. Neural adaptive video streaming with pensieve. In Proceedings of the 2017 ACM SIGCOMM Conference (2017), ACM.

Neely, Michael J. Stochastic network optimization with application to commu- nication and queueing systems. Synthesis Lectures on Communication Networks 3, 1 (2010), 1–211.

Neely, Michael J. Dynamic optimization and learning for renewal systems. IEEE Transactions on Automatic Control 58, 1 (2012), 32–46.

Niamut, Omar A, Thomas, Emmanuel, D'Acunto, Lucia, Concolato, Cyril, De- noual, Franck, and Lim, Seong Yong. MPEG DASH SRD: spatial relationship description. In Proceedings of the 7th International Conference on Multimedia Systems (2016), ACM.

Palmer, Mirko, Appel, Malte, Spiteri, Kevin, Chandrasekaran, Balakrishnan, Feldmann, Anja, and Sitaraman, Ramesh. Video streaming optimization across enriched layers, 2020. Unpublished.

Palmer, Mirko, Kru¨ger, Thorben, Chandrasekaran, Balakrishnan, and Feldmann, Anja. The quic fix for optimal video streaming. In Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC (New York, NY, USA, 2018), EPIQ'18, ACM, pp. 43–49.

Park, Jounsup, and Nahrstedt, Klara. Navigation graph for tiled media stream- ing. In Proceedings of the 27th ACM International Conference on Multimedia (New York, NY, USA, 2019), MM '19, ACM, pp. 447–455.

Petrangeli, Stefano, Swaminathan, Viswanathan, Hosseini, Mohammad, and De Turck, Filip. An HTTP/2-based adaptive streaming framework for 360 vir- tual reality videos. In Proceedings of the 25th ACM international conference on Multimedia (2017), pp. 306–314.

Riiser, Haakon, Vigmostad, Paul, Griwodz, Carsten, and Halvorsen, P˚al. Com- mute path bandwidth traces from 3G networks: analysis and applications. In Proceedings of the 4th ACM Multimedia Systems Conference (2013), ACM, pp. 114–118.

Romaniak, Piotr, and Janowski, Lucjan. How to build an objective model for packet loss effect on high definition content based on ssim and subjective ex- periments. In Future Multimedia Networking (Berlin, Heidelberg, 2010), Sher- ali Zeadally, Eduardo Cerqueira, Mar´ılia Curado, and Miko-laj Leszczuk, Eds., Springer Berlin Heidelberg, pp. 46–56.

Romirer-Maierhofer, Peter, Ricciato, Fabio, D'Alconzo, Alessandro, Franzan, Robert, and Karner, Wolfgang. Network-wide measurements of TCP RTT in 3G. In Traffic Monitoring and Analysis. Springer Berlin Heidelberg, 2009, pp. 17–25.

Seufert, Michael, Egger, Sebastian, Slanina, Martin, Zinner, Thomas, Hoßfeld, Tobias, and Tran-Gia, Phuoc. A survey on quality of experience of http adaptive streaming. IEEE Communications Surveys & Tutorials 17, 1 (2014), 469–492.

Sitaraman, Ramesh K. Network performance: Does it really matter to users and by how much? In Proc. COMSNETS (2013).

Spiteri, Kevin, Sitaraman, Ramesh, and Sparacio, Daniel. From theory to prac- tice: Improving bitrate adaptation in the DASH reference player. ACM Trans. Multimedia Comput. Commun. Appl. 15, 2s (July 2019).

Spiteri, Kevin, Urgaonkar, Rahul, and Sitaraman, Ramesh K. BOLA: near- optimal bitrate adaptation for online videos. IEEE/ACM Transactions on Networking 28, 4 (2020), 1698–1711.

Stockhammer, Thomas. Dynamic Adaptive Streaming over HTTP – Standards and Design Principles. In Proc. ACM MMSys (2011).

Stohr, Denny, Fr¨ommgen, Alexander, Rizk, Amr, Zink, Michael, Steinmetz, Ralf, and Effelsberg, Wolfgang. Where are the sweet spots? a systematic approach to reproducible dash player comparisons. In Proceedings of the 2017 ACM on Multimedia Conference (2017), ACM.

Vandalore, Bobby, Feng, Wu-chi, Jain, Raj, and Fahmy, Sonia. A survey of application layer techniques for adaptive streaming of multimedia. Real-Time Imaging 7, 3 (2001), 221–235.

Wang, Zhou, Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image Quality Assessment: From Error Visibility to Structural Similarity. Trans. Img. Proc. 13, 4 (Apr. 2004).

Wu, Chenglei, Tan, Zhihao, Wang, Zhi, and Yang, Shiqiang. A dataset for exploring user behaviors in VR spherical video streaming. In Proceedings of the 8th ACM on Multimedia Systems Conference (2017), pp. 193–198.

Yahia, Mariem Ben, Louedec, Yannick Le, Simon, Gwendal, Nuaymi, Loutfi, and Corbillon, Xavier. HTTP/2-based frame discarding for low-latency adaptive video streaming. ACM Trans. Multimedia Comput. Commun. Appl. 15, 1 (Feb. 2019), 18:1–18:23.

Yin, Xiaoqi, Jindal, Abhishek, Sekar, Vyas, and Sinopoli, Bruno. A control- theoretic approach for dynamic adaptive video streaming over HTTP. In Proc. ACM SIGCOMM (2015).

Zambelli, Alex. IIS smooth streaming technical overview. Microsoft Corporation. (2009).

Zink, Michael, Sitaraman, Ramesh, and Nahrstedt, Klara. Scalable 360° video stream delivery: Challenges, solutions, and opportunities. Proceedings of the IEEE 107, 4 (2019), 639–650.

Algoritma Video streaming

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Bio Data Penulis

Penulis lahir di Semarang pada tanggal 1 Maret 1983. Penulis menempuh pendidikan Sarjana Teknik Elektro di Universitas Kristen Satya Wacana (UKSW), lulus tahun 2004, kemudian tahun 2005 melanjutkan studi pada Magister Desain pada Fakultas Seni Rupa dan Desain, Institut Teknologi Bandung (ITB), dan melanjutkan studi pada program studi Teknologi Multimedia pada Swinburne University of Technology Australia.

Penulis sejak tahun 2010, menjadi dosen pada program studi Desain Grafis Universitas Sains dan Teknologi Komputer (Universitas STEKOM), memiliki jabatan fungsional Lektor 300 dan sedang proses mengajukan kenaikan jabatan fungsional menjadi Lektor Kepala. Penulis juga seorang wirausaha di bidang toko online yang berhasil di kota Semarang dan juga aktif sebagai freelancer dalam bidang fotografi, web design dan multimedia.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
AYASAN PRIMA AGUS TEKNIK

Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8120-54-3 (PDF)

9 786238 120543