NTNU – Trondheim
Norwegian University of
Science and Technology

# P2P Video Streaming with HTML5 and WebRTC

**Martin Kirkholt Melhus**

Master of Science in Communication Technology
Submission date: June 2015
Supervisor: Otto Wittner, ITEM
Co-supervisor: Poul Einar Heegaard, ITEM

Norwegian University of Science and Technology
Department of Telematics

|  |  |
|---|---|
| **Title:** | P2P Video Streaming with HTML5 and WebRTC |
| **Student:** | Martin Kirkholt Melhus |

**Problem description:**

P2P architectures for video streaming have been shown to fulfill the requirements of live video streaming, but earlier systems required specialized clients or browser plug-ins. The emerging WebRTC standardization work allow for native P2P applications within modern web browsers.

The goal of this project is to evaluate schemes for single source live P2P video streaming and adapt one or more of these schemes into a prototype application using WebRTC. The criteria for the prototype will be to minimize cost in terms of bandwidth for the content provider, while providing a video stream with acceptable quality for peers in an environment subject to churn and heterogeneous clients.

Novel ideas to make existing designs cohere with the additional components of WebRTC should be presented. Such ideas should seek to manipulate and optimize the topology of the P2P network.

Required work can be broken down into the segments:

- Collect and study related work.
- Design a system for video streaming in the WebRTC domain using the knowledge obtained.
- Give an account of the trade-offs and key parameters introduced by the design choices.
- Implement the system, to the extent that time allows.

|  |  |
|---|---|
| **Responsible professor:** | Otto Wittner, NTNU |
| **Supervisor:** | Poul E. Heegaard, NTNU |

## Abstract

WebRTC provide the possibility of establishing peer-to-peer connections within a web browser, without requiring plug-ins. This suggests that peer-to-peer technology that was previously only available through specialized software is now available in a browser.

This thesis presents Dugnad; a scalable live video streaming architecture designed for WebRTC. In the core of the architecture is use of virtual coordinates to push data in implicit directed acyclic graphs. Challenges and opportunities related to the architecture and the WebRTC platform are identified and discussed.

Experimentation with a simple implementation of Dugnad is conducted, and as the presented results identify some issues with the implementation, improvements are suggested. The results show that WebRTC is a capable building block for scalable live video streaming within a web browser.

## Sammendrag

WebRTC muliggjør opprettelse av kommunikasjonskanaler direkte mellom nettlesere uten å kreve at brukeren installerer tilleggsprogrammer. Dette tilsier at teknologier som baserer seg på jevnbyrdsnett, og som tidligere krevde spesialisert programvare, nå kan benyttes i nettleseren.

Denne avhandlingen presenterer Dugnad, en arkitektur for skalerbar direktesendt video-strømming, som bygger på WebRTC. Sentralt i arkitekturen ligger bruk av virtuelle koordinater for å dytte data i en implisitt rettet asyklisk graf. Utfordringer og muligheter relatert til arkitekturen, og til plattformen skapt av WebRTC, blir presentert og drøftet.

Resultatene av forsøk med en enkel implementasjon av Dugnad blir presentert, og resultatene fremhever svakheter med denne implementasjonen. Basert på disse blir det foreslått forbedringer til både implementasjonen og selve arkitekturen. Det går frem av resultatene at WebRTC er et egnet grunnlag for å bygge applikasjoner for direktesendt video-strømming i en nettleser.

# Preface

This thesis marks the conclusion of my five years at the master's program in communication technology at NTNU. During this period I was as able to spend one year at EURECOM, where my fascination for peer-to-peer video streaming was born. Had I not attended Prof. Ernst W. Biersack's course on Internet Applications, this thesis would probably discuss a completely different topic.

I would like to thank my supervisors, Otto Wittner and Poul E. Heegaard, who have provided valuable suggestions and feedback throughout the whole process of writing this thesis. Also, I must extend my thanks to Tarjei Husøy who helped me cover my blind spots and help me to maintain an acceptable quality of writing throughout this thesis.

# Contents

# List of Figures

# List of Tables

# List of Listings

# List of Acronyms

**API** Application Program Interface.

**CDN** Content Delivery Network.

**DAG** Directed Acyclic Graph.

**EED** End-to-End Delay.

**HTML** Hypertext Markup Language.

**HTTP** Hypertext Transfer Protocol.

**ICE** Interactive Connectivity Establishment.

**IP** Internet Protocol.

**IPTV** Internet Protocol Television.

**ISP** Internet Service Provider.

**JSEP** JavaScript Session Establishment Protocol.

**JSON** JavaScript Object Notation.

**NAT** Network Address Translation.

**P2P** Peer-to-Peer.

**PDU** Protocol Data Unit.

**RP** Rendezvous Point.

**RP/LB** Random Peer, Latest Blind Chunk.

**RTT** Round-Trip Time.

|  |  |
|---|---|
| **STUN** | Session Traversal Utilities for NAT. |
| **TCP** | Transmission Control Protocol. |
| **TURN** | Traversal Using Relays around NAT. |
| **UDP** | User Datagram Protocol. |
| **VOD** | Video on Demand. |
| **W3C** | World Wide Web Consortium. |
| **WebRTC** | Web Real-Time Communications. |

# Chapter 1 Introduction

## 1.1 Motivation

This thesis is motivated by previous research on live video streaming using Peer-to-Peer (P2P) technologies and their proven scalability. With the new features provided by Web Real-Time Communications (WebRTC), these technologies are made available inside browsers. Plug-ins are no longer required, as P2P connections are natively supported by modern web browsers through WebRTC. Applying P2P schemes can reduce the cost for a broadcaster in terms of required bandwidth and network infrastructure. This can ultimately reduce the monetary cost of providing live video broadcasting with a satisfactory user experience.

## 1.2 Problem and Scope

The goal of this thesis is to evaluate schemes for scalable single-source live video streaming that can be run without plug-ins by modern web browsers. The thesis looks at proposed overlay networks for video streaming, and relates which parts are compatible with the WebRTC domain. Then, it describes Dugnad, an architecture featuring an overlay network that takes advantage of the components of WebRTC.

The original problem description proposes to account for trade-offs prior to implementing the system. However, understanding the parameters of the system was in some cases found to either require a simulation model of the system or experimentation on an implemented version. As a result, a lot of time was used on the experimental implementation, as it was used for this purpose.

The scope of the implementation is limited to providing a broadcast data channel for timely delivery of chunks of binary data; issues relating to video encoding are outside the scope of this thesis. Acceptable quality of a video stream, as mentioned in the problem description, has been interpreted as the ability to broadcast a stream of data with equal bit rate to a high-definition video stream with a short delay. The research conducted is focused on single-source live video streaming, hence studying systems providing P2P-based Video on Demand (VOD) is outside the scope of the thesis.

While some security considerations are discussed, this thesis does not provide a detailed study regarding the security of Dugnad. Copyright issues relating to broadcasting media are prevalent, but also outside the scope of this thesis.

## 1.3 Methodology

A substantial part of this thesis is studying existing overlay networks and P2P video streaming architectures. The result of the study is summarized in a list of desired properties for a P2P video streaming architecture for WebRTC.

An architecture based on the desired properties is designed, and simple versions of the components in the architecture are implemented. Small-scale experiments with roughly 100 nodes are conducted, while the architecture aims to support thousands. Based on results of the small-scale experiments, combined with insight from the conducted research on P2P video streaming architectures, suggestions to improvements of both the implementation and the proposed architecture are given.

## 1.4 Contribution

This thesis presents an architecture for P2P live video streaming in modern web browsers. Small-scale performance measurements are conducted, and the results are positive regarding WebRTC's potential to provide a platform for efficient P2P based broadcasting of high-definition video streams.

## 1.5 Outline

Chapter 2 gives an overview of P2P technologies and protocols that have inspired Dugnad. In chapter 3, a detailed account of some key aspects of previous studies are given, and their application in the WebRTC domain is discussed. The chapter concludes with a list of desired properties for Dugnad. Chapter 4 gives a detailed description of the Dugnad protocol, while chapter 5 accounts for the decisions made and parameters chosen in the implementation. The experiment setup and the obtained results are described in chapter 6. Chapter 7 includes a discussion of the results as well as other aspects of the Dugnad architecture and P2P live video streaming. Chapter 8 adds concluding remarks and suggests relevant future work.

# Chapter 2 Background

This chapter outlines a few relevant protocols related to P2P live video streaming. The protocols are categorized and differences are identified.

## 2.1 Overlay Networks

Overlay networks are networks built on top of other networks such as the Internet. In P2P systems, virtual links between peers are made, allowing for an abstraction of the underlying network and the construction of distributed systems.

**Pastry** Pastry is an overlay network and a generic object location and routing scheme used in a range of P2P applications [RD01]. In Pastry, all nodes have random unique numeric identifiers (nodeId). Nodes are aware of a set of other nodes (peers) and their associated nodeId. Messages contain a numeric key; in the routing step a node forwards the message to a peer for which nodeId is numerically closest to the key contained in the message.

## 2.2 Structured Overlay Networks

A naive way of implementing a multicast overlay network is to position arriving nodes in a tree-structure where the broadcaster is the root node. Interior nodes forward the media data to their children; hence full diffusion of a chunk is achieved when all leaf nodes have received the chunk. This system is very simple when there is no node churn, yet it has some caveats. Only interior nodes will participate in the diffusion scheme, hence for a binary tree structure, half of the nodes are leaf nodes and will carry no forwarding load. Additional problems arise when the tree is subject to node churn, and the overlay network, i.e. the tree property must be maintained.

**SplitStream** SplitStream is a scheme that enables high-bandwidth multicast in P2P networks [CDK+03]. To address the problem that in tree-based architectures leaf nodes carry none of the forwarding load of multicast messages, SplitStream utilizes a forest of interior-node-disjoint multicast trees. This way, a node can be both an interior node and a leaf node, as it is part of multiple trees. Splitting the content into multiple stripes to be forwarded into separate multicast trees provide redundancy, and thus tolerance to churn. If a node fails, it should on average only cause one stripe to fail, since the node is only an internal node in a single tree.

## 2.3 Swarming-Based Overlay Networks

Unstructured overlay networks can be used in diffusion schemes for live media streaming. Gossip protocols are an important part of such systems, and they are based on the following mechanisms [SHM07].

**Pull** In a pull based system, a user will select a piece it does not already possess and request it from a target peer.

**Push** In a push based system, a user will select a piece it possesses and transmit it to the target.

Protocols can also be interleaved, meaning that they are combining the push and pull mechanisms. Research suggests that interleaved protocols are better suited for disseminating a series of pieces, e.g. media streams, than protocols relying only on either push or pull [SHM07].

An example of a simple push based-gossip protocol is the Random Peer, Latest Blind Chunk (RP/LB). A node selects a random peer to whom he transfers blindly the available chunk with the highest sequence number. The diffusion rate of this protocol is very poor, as shown in [BMM+08].

**DP/LU** An improved general push-based gossip protocol is the Deprived Peer, Latest Useful Chunk (RP/LB). Instead of selecting receivers at random, a peer will select its most deprived peer, i.e. the peer who has the least amount of viable chunks. The chunk that is pushed is the latest useful chunk, being the most recent chunk that is not possessed by the deprived peer receiver. This protocol requires exchange of availability information to enable the selection of a latest useful chunk. Research suggests that RP/LB is as a well suited protocol for live streaming implementations [BMM+08] .

**PPLive** PPLive is the name of a mesh-pull protocol for P2P Internet Protocol Television (IPTV). Despite being proprietary, some parts of the protocol has been uncovered [SGGS09, HLL$^+$07]. Peers are identified by their Internet Protocol (IP)-address and port number, and the platform consists of multiple overlays, each corresponding to a channel. A *channel list* containing information about available channels can be downloaded over Hypertext Transfer Protocol (HTTP) from a *channel list server*. Within an overlay, peers query a *membership server* over User Datagram Protocol (UDP) to collect a set of peers, which are again contacted to obtain an aggregate list of peers. This way a peer will obtain an overview of the other peers tuned in to the same channel.

**CoolStreaming/DONet** CoolStreaming is an implementation of DONet, a Data-driven Overlay Network for live media streaming [ZLLY05]. The core characteristic of DONet is an exchange of data availability information. Based on this information, a node will retrieve unavailable data from its set of partners, as well as supplying available data through pull actions. The architecture is churn tolerant, as the set of partners is periodically updated. Directions as enforced by a structured architecture do not apply in DONet, as it is only driven by availability.

**GridMedia** Gridmedia is a P2P-based live video streaming system which uses a push-pull diffusion scheme [ZLZ$^+$05]. It also uses a static well-known Rendezvous Point (RP) to assist in bootstrapping arriving nodes, requiring the RP to have a partial view of the overlay network. Information about all nodes in the overlay network is recorded at the RP, allowing nodes to obtain candidate partner lists. When a node receives the candidate list, it measures the End-to-End Delay (EED) to the candidate, and is more likely to pick nodes with smaller EEDs. The RP is able to utilize its partial view to ensure existing nodes with long matching IP address prefixes are included in the arriving node's partner candidates. For nodes in the same subnet, or behind the same Network Address Translation (NAT) router, this enables connections with short network distances, and thus low EEDs.

## 2.4 Protocol Comparison

A comparison including some key differences between the mentioned protocols is given in Table 2.1.

## 2.5 WebRTC

WebRTC is a World Wide Web Consortium (W3C) working draft [BBJ15]. It defines a set of Application Program Interfaces (APIs) that allow for real-time communication between browsers that implement them. This includes the ability to send media streams and other binary data in a P2P manner between clients, as well as providing

| Protocol | Overlay | Peer Selection | Diffusion Mechanism |
|---|---|---|---|
| SplitStream | Structured | Deterministic | Push |
| DP/LU | Swarming | Availability driven | Push |
| CoolStreaming | Swarming | Availability driven | Pull |
| GridMedia | Swarming | Availability driven (pull)<br>Resource aware (push) | Interleaved |
| PPLive | Swarming | - | Interleaved |

Table 2.1: A comparison of different architectures for Peer-to-Peer (P2P) live video streaming.

clients with the possibility of accessing local media devices from within the browser sandbox. Further details regarding WebRTC are provided in the following chapter.

# Chapter 3
# Evaluation of Streaming Architectures

The following chapter aim to relate properties of existing solutions for P2P live video streaming to their viability in the WebRTC domain. Based on the findings, a list of desired properties for a WebRTC based live streaming protocol is proposed.

## 3.1 Broadcast Strategies

The following section will describe different strategies for broadcasting data and look at their scalability.

### 3.1.1 Unicast

The traditional implementation of video streaming solutions, consists of dedicated source nodes serving unicast media streams to individual nodes. These architectures are not scalable in terms of bandwidth requirements for source nodes. In a system with a single source node, the required upload bandwidth $b$ grows linearly with the number of nodes $n$ in the system, i.e. $b = \mathcal{O}(n)$, as seen in Figure 3.1. On the other hand, these systems are churn tolerant, as churn will not affect the system in any other way than freeing up bandwidth at the broadcaster.

Additional source nodes can be included to be able to handle bigger loads, but the required capacity still grows linearly, which makes this an expensive solution. As a result, unicast systems tend to rely on Content Delivery Networks (CDNs) to achieve sufficient capacity.

In WebRTC, unicast media streams are supported through the `MediaStream` API [ALMC14].

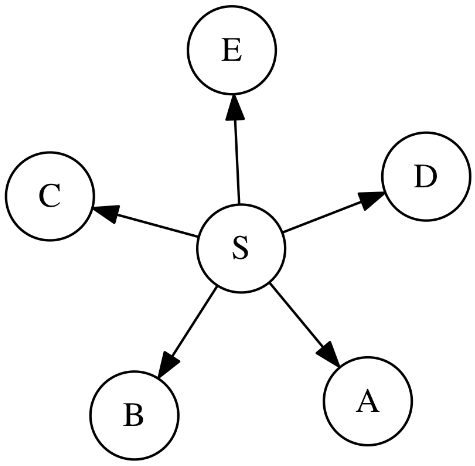


Figure 3.1: In a unicast system, the source node $S$ serves a stream of data at a rate $r$ the other $n$ nodes in the overlay network. This requires an upload bandwidth $b_S = rn$.

### 3.1.2 Network Layer Multicast

Network layer multicast is supported through the IP multicast extension [Dee89]. A sender is only required to send a single copy of each datagram, and compatible multicast routers duplicate and forward the datagram such that all members of a destination group receive it. The service is best effort, meaning that in terms of reliability, the service is equal to that of regular unicast over IP.

All routers between the source and all destinations must be multicast compatible for network layer multicast to work. Due to limited deployment of compatible network infrastructure, IP multicast is not suitable for live video streaming at a global scale [CDK+03, ZLZ+05].

### 3.1.3 Peer-to-Peer

An alternative to traditional unicast systems is to rely on P2P strategies. This requires nodes to be capable of communicating directly with other nodes, such that the source node needs only to be connected to a subset of nodes in the network. Nodes use their communication capabilities to help forward the media stream between each other, and the required forwarding load of the source node is lowered due to load distribution.

In Figure 3.2 the source node $S$ can broadcast chunks of data to all nodes in the network by sending a single copy of each chunk to node $B$ and relying on this node to forward the data to subsequent nodes. However, this approach is dependent on node $B$'s bandwidth and willingness to cooperate. If B's upload bandwidth $b_B$ is lower than that of the source node $b_S$ i.e. $b_B < b_S$, strictly relying on $B$ to forward a stream of chunks to all other nodes will fail due to lack of bandwidth and an ever-growing output buffer. The probability of reaching all nodes, as well as the rate at which this is achieved for individual chunks can be improved by duplicating the data. The fanout of the source node $k$ is the number of nodes that receive chunks directly from the source node.

The theoretical upload bandwidth requirement of the source node does not grow with the number of nodes in the overlay network, as it is limited by the chosen constant fanout $k$ of the source node, hence $b = \mathcal{O}(1)$. To logically connect the source node to all other nodes, $k$ needs to be equal to the number of edges from the source in a spanning tree containing all nodes in the overlay network. However, since connections can be made between arbitrary nodes, the spanning tree can be constructed from a complete graph. This suggests that a spanning tree can always be made with a single edge to the source node, hence $k = 1$ is sufficient to logically connect a source node to all other nodes in the network. Note that $k = 1$ can be too small to sustain a stream of chunks at rate $r$ to all peers due to individual nodes bandwidth limitations.

P2P-architectures need to address the problem of churn and maintaining healthy peer sets, i.e. knowledge of other nodes in the overlay network. This problem can be approached in different ways, depending on whether the P2P overlay network is structured or swarming-based. However, mechanisms for providing tolerance to churn require signaling and ultimately requires some additional bandwidth.

In WebRTC, establishment of data channels between nodes is supported through the `DataChannel` API, allowing nodes to exchange data in a P2P manner.

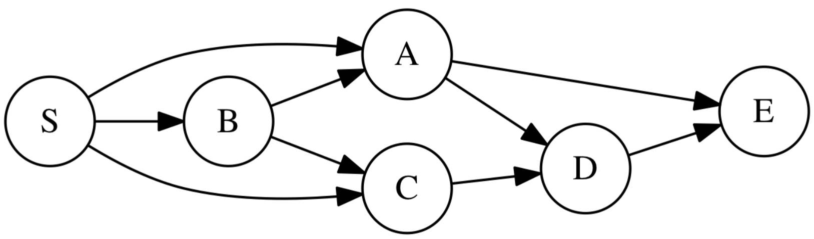


Figure 3.2: In a generic Peer-to-Peer (P2P) system, the source node $S$ serves a limited number of nodes $k$ a stream of data at rate $r$. The total number of nodes save the source node is $n$. The required upload bandwidth $b_S$ for $S$ is $b_S = r \times min(k, n)$.

## 3.2 Overview of the WebRTC Domain

Traditional P2P overlay architectures rely in the fact that nodes have public IP-addresses and are actively listening on a specified port. When putting aside the complications introduced by firewalls and NAT, this enables negotiation of new connections directly based on the knowledge of another node's public IP address.

In WebRTC, a `PeerConnection` can be established between users to allow browser-to-browser (P2P) communication [RTCb]. However, for this to be possible, a signaling channel is required, and `WebSockets` allow this channel to be provided by the web server [Hic12, ALMC14]. A PeerConnection can contain a `DataChannel` as discussed in subsection 3.1.3, representing a bidirectional data channel between peers [RTCa].

A session between the peers is established using JavaScript Session Establishment Protocol (JSEP). JSEP provides mechanisms for creating offers and answers [ALMC14], thus defining the contents of the required session negotiation messages. Note that how these messages are exchanged between two potential peers is not specified in JSEP, but is made possible by WebSocket connections between browsers and the Web server [VWS13]. In Figure 3.3 the different communication channels in a WebRTC architecture are outlined, where the media path denotes a `DataChannel` between the two parties.

## 3.3 Limitations Imposed by WebRTC

This section identifies some key differences between traditional P2P applications and P2P applications running in a web browser. Additionally, limitations relating to NAT traversal are outlined.

### 3.3.1 Connection Establishment

A key difference between a traditional P2P applications and a WebRTC web application is the inability to directly establish connections to peers, despite knowing their public IP-address. Restrictions imposed by the browser sandbox disallow a JavaScript application to listen for incoming traffic from arbitrary sources. Thus to be able to exchange P2P messages in WebRTC, clients need to negotiate data channels using an existing signaling channel. If the goal is to send chunks of binary data, a `DataChannel` must be negotiated between the peers [ALMC14].

A Web application using WebRTC will be loaded from a web server. By allowing the clients to connect to the web server using WebSockets, it can relay messages back and forth between clients, and thus supply the required signaling channel, as seen in Figure 3.3. Hence the web server is able to act as a proxy for JSEP signaling,

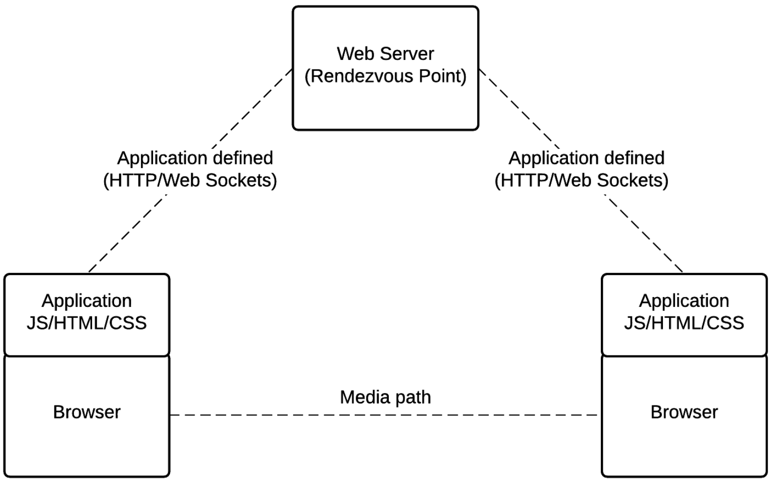


Figure 3.3: Data paths in a Web Real-Time Communications (WebRTC) architecture based on Figure 2 in [Alv14]. Session descriptions are negotiated using WebSockets with the Web Server acting as a proxy.

allowing connection establishment between clients, thus acting as the RP in the overlay network.

### 3.3.2 NAT Traversal

Additional components are required by WebRTC to enable NAT traversal. These components are Session Traversal Utilities for NAT (STUN) servers and Traversal Using Relays around NAT (TURN) servers, and are required for clients located behind NATs to locate each other [ALMC14].

STUN does not impact the way data flows between clients after a connection has been established, as it only helps a host to discover its own public IP address, enabling NAT traversal [RMMW08]. However, STUN does not always solve the problem, as the obtained addresses may not work in all cases. For this reason, TURN servers are used when STUN fails. A TURN server will relay the data between two peers; hence data has to flow through the relay server [MMR10].

In 2014, an analysis based on 6 million minutes of video calls from a WebRTC application vendor showed that 92% of the calls were real P2P without using a relay.[1] This means that only 8% of the calls required a TURN server for NAT traversal. Incorporation of a TURN server has been omitted in the experiments conducted as part of this thesis.

## 3.4 Overlay Network Topology

This section looks at some properties of the two types of overlay networks.

### 3.4.1 Structured Overlay Networks

Structured systems for P2P media streaming have traditionally meant the construction of trees with unidirectional data flows. An example is CoolStreaming, which is briefly discussed in section 2.3. These systems have been studied with analytical models, but a clear understanding of the underlying performance trade-offs is lacking [BMM+08].

A notable property of tree-based networks is the absence of cyclic data flows, i.e. bandwidth waste due to chunks being sent to nodes that already possess the data. In a tree-structured overlay network, a parent can safely push data to child nodes, knowing that they will not already possess the data. Hence if $A$ and $B$ are nodes in a tree where $B$ is a child of $A$, $A$ cannot receive a chunk from $B$ later than it was received at $B$ since the tree property prevents $B$ from being an ancestor of $A$.

### 3.4.2 Swarming Based Overlay Networks

Elaborate schemes are involved in constructing effective structured overlay networks such as in SplitStream [CDK+03], and are not made easier by the fact that WebRTC prevents direct communication between nodes prior to connection establishment. In addition to maintaining a tree property in the overlay network being a demanding task under heavy churn, internal nodes can become bottlenecks for the subtree they serve [PPKB07]. This suggests that swarming-based overlay networks are better suited for the WebRTC domain.

Gridmedia takes advantage of a centralized component acting as a RP for arriving nodes. This approach is compatible with WebRTC; the RP in Gridmedia records information about nodes in the overlay network and supply necessary information contacting nodes in the overlay to arriving nodes. Having a RP with this responsibility is required in WebRTC, as discussed in subsection 3.3.1.

[1] The results are available at http://webrtcstats.com/webrtc-revolution-in-progress/. (Accessed April 29. 2015). A similar study, of 1,500,000 minutes of video calls in 2013, reported the amount of relayed calls to 14%. These results are available at http://webrtcstats.com/first-webrtc-statistics/ (Accessed April 29. 2015).

The RP in Gridmedia uses IP prefix matching as a criterion for selecting some of the nodes proposed in the candidate list. This allows the RP to manipulate what connections are made with the intention of using its knowledge of the nodes to help construction of an efficient overlay network. Unfortunately, prefix matches in IP addresses do not accurately describe the network distance between two nodes. Note that in WebRTC, it is not possible for an arriving node to measure the EED to nodes contained in the candidate list provided by the RP prior to establishing a connection. Since establishing a connection means signaling exchanges using the RP as a proxy, measuring EEDs to all potential peers is not feasible in the WebRTC domain.

## 3.5 Peer/Chunk Selection Policies

A peer selection policy describes how a node selects a peer to which it will transfer data. The chunk selection policy denotes what data the node will transfer. These are not necessarily independent of each other, as the selection of a peer may depend on what chunk the node intends to send. Likewise, if the peer is selected prior to the chunk, this may affect which chunk should be transferred.

There are two approaches to peer selection: an agnostic approach, where peers are chosen at random; and a resource aware approach, where properties of the peer such as availability of data or bandwidth capacity are weighted into the decision.

In the WebRTC domain, finding the available bandwidth capacity of a peer is not trivial. The source node is not downloading data; hence measuring the download capacity of peers is impossible rendering Tit-for-Tat schemes useless for the first hop. Identifying this problem, [MP10] suggests that providing a high enough source capacity that is capable of uploading a sufficient number of copies of every chunk, an agnostic peer selection in the source node should perform well. Furthermore, it is suggested that applying resource awareness to some degree, while keeping elements of randomness in the peer selection policy can outperform agnostic peer selection. However pure resource aware selection policy cause discrimination of nodes with fewer resources, thus causing high diffusion delays for these nodes. A premise of any resource aware selection policy is possessing information about available resources of peers. Thus signaling is required, as this information must be exchanged regularly since it changes over time.

An availability-driven peer selection policy, as the one found in pull-based protocol DONet could introduce large delays. In DONet, at least three EEDs are required for a chunk to be transferred a single hop, hence the aggregate delay in a large overlay network is substantial [ZLZ$^+$05].

## 3.6 Desired Properties of a Live Streaming architecture

The following list of desired properties of a WebRTC framework for live media streaming is deduced from the protocols studied in this chapter and in chapter 2.

1. Scalable
    a) Bandwidth required for forwarding media data by the source node should not grow with the number of connected nodes. This suggests a P2P-based system where required upload bandwidth is $\mathcal{O}(1)$ of the number of nodes, as discussed in subsection 3.1.3.
    b) Bandwidth required for signaling should not grow with the number of nodes in the network.
2. Connection Establishment
    a) WebRTC requires the web server to act as a proxy for connection establishments by having WebSocket connections to clients as discussed in section 3.2. Thus the web server should act as a Rendezvous Point (RP), allowing clients to join the overlay network.
    b) The RP should use its knowledge of the overlay network to construct the list of candidate partners based on properties of arriving nodes.
    c) The system should allow users behind NAT access by providing means for NAT traversal. This is solved using STUN and TURN which can be provided by a third party.
3. Overlay Network
    a) The overlay network should be swarming-based for reduced complexity, as discussed in section 3.4
    b) The overlay network should include mechanisms preventing cyclic data flows as discussed in subsection 3.4.1
    c) The overlay network should motivate connection establishment between nodes with short network distances.
4. Peer Selection
    a) The peer selection policy should apply some degree of resource awareness as discussed in section 3.5.
5. Diffusion Mechanism
    a) The diffusion scheme should be interleaved, i.e. combining push and pull gossiping for reasons discussed in section 2.3 and in section 3.5.

Chapter 4

# Dugnad: A P2P Media Streaming Architecture for WebRTC

This chapter will explore an approach to content distribution based on self-organizing Directed Acyclic Graphs (DAGs), designed for the WebRTC domain. The name Dugnad is taken from the Norwegian term for people gathering to performing voluntary work as part of a community.

## 4.1 Architecture

In this section, the components of the Dugnad architecture and how they interact is described.

### 4.1.1 Components

The Dugnad architecture is composed of several components.

**Rendezvous Point (RP)** The web server who serves the index web page, as well as communicating with connected hosts using WebSockets. Additionally this server acts as a proxy between web clients during the connection establishment phase. The RP is not cooperating in the diffusion scheme for media data.

**Broadcaster** A single node in the network that produce a video stream to send to all the other nodes in the network.

**Viewer** All nodes in the network except the broadcaster.

**Broadcast Channel** A broadcast channel represents overlay network[s] dedicated to disseminate a media stream from a broadcaster. It is identified by a *broadcast channel identifier*. Every broadcaster has a single unique broadcast channel identifier. Broadcasters providing streams of different rates will participate in multiple overlay networks (one per unique stream), under the same channel identifier.

Note that when describing Dugnad, the term *node* is used when referring to either a broadcaster or a viewer.

### 4.1.2 Client-Server Handshake

After loading the application from the web server, the client, i.e. the viewer or broadcaster, perform a handshake with the RP to establish a WebSocket connection.

The broadcaster will initiate this handshake with a `create` message, containing a broadcast channel identifier in addition to information about the provided streaming qualities and other relevant information about the media stream. The server will answer with a `created` message if the broadcaster is authorized to initialize the broadcast channel.

A viewer will initiate the handshake with a `join` message. The message should contain information about the viewer's bandwidth capacity in terms of bit rates as well as supported codecs. If the broadcaster is providing a stream that is compatible with the viewers requirements, the server will associate the viewer with the overlay network in the broadcast channel representing that stream.

Since the handshake associates a client to a unique overlay network and a broadcast channel, a single web server can act as a RP for multiple broadcast channels.

### 4.1.3 Overlay Network Bootstrap

The bootstrap procedure follows the Client-Server Handshake for all viewers, and its purpose is to establish connections between arriving viewers and existing nodes in the overlay network. The procedure involves obtaining a list of candidate peers, and to establish connections to a selection of these candidates with the help of the RP. The sequence of messages making up the bootstrap procedure is depicted in Figure 4.1 and detailed description of each of the exchanged messages is given in Table 4.1.

After completion of the bootstrap procedure, a `PeerConnection` containing a `DataChannel` is established between the arriving viewer and one or more nodes contained in the obtained peer candidates.

## 4.2 Protocol Data Units

The following section will describe the structure of the Protocol Data Units (PDUs) used in the broadcasting protocol.

| # | Type | Description |
|---|---|---|
| 1 | `bootstrap` | The initial message is sent from the arriving node to the RP. The message has no required body other than the message subject, which should be `bootstrap`. |
| 2 | `channel list` | The response to the arriving node's bootstrap request comes in a message with the subject `boostrap_channels`. The data is a JSON formatted string containing several entries. First, the `coordinate` field indicates the arriving node's assigned coordinate. Additionally there are two lists of potential peers: the `parents` and `children` lists containing the coordinate of potential nodes as well as a `channel_id` which can be used to send an `offer` to the individual nodes. |
| 3 | `offer` | The offer is contains session description data that indicates that the node wants to open a `PeerConnection` with a `RTCDataChannel`. Additionally it contains a `channel_id` derived from the data received in message 2, which allows the RP to find the correct recipient for the offer. |
| 4 | `offer` | The offer from the arriving node is forwarded to the recipient based on the `channel_id`. |
| 5 | `answer` | The node which received the offer, sends the answer back to the RP, with the same `channel_id` which was used in the offer, thus allowing the RP to find the correct initiator of the request. |
| 6 | `answer` | The answer is forwarded back to the arriving node based on the `channel_id`. |
| 7 | `data` | The connection with the `DataChannel` is now established and the nodes are able to send data directly to each other. |

Table 4.1: Messages exchanged in the bootstrap procedure.

### 4.2.1 Communication Channel Constraints

The `DataChannel` objects that are used to send data between peers is capable of sending `String` and `ArrayBuffer` objects. In the suggested implementation, `ArrayBuffer` is used when sending media data, while `String` objects are used for signaling with a JavaScript Object Notation (JSON) serialized payload.

### 4.2.2 Binary Representation of Media Streams

A media stream is split into *chunks* representing several seconds of a media stream. Chunks contain an identifier (`chunk_num`), which indicate where they belong in the original sequence of chunks produced by the broadcaster. The media contained in

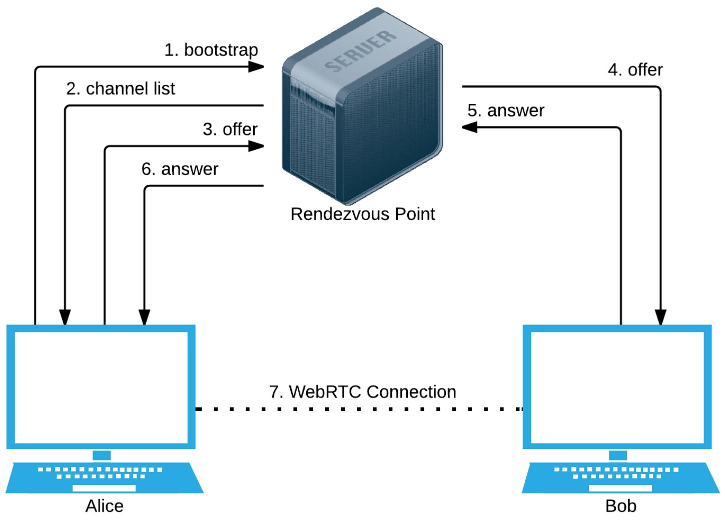


Figure 4.1: Bob is watching a stream and Alice wants to join. Alice contacts the Rendezvous Point (RP) to establish connections to nodes in the overlay network. First, Alice sends a bootstrap request (1) for which the RP responds with a channel list (2) containing signaling channel identifiers that can be used to contact other nodes. Alice selects a `channel_id` contained in the channel list and sends an offer (3), which the RP forwards to the preselected receiver for that channel (4) which is Bob. Bob wants to accept the offer, thus he sends an answer to the RP using the `channel_id` used for the offer (5). The resolves the receiver of the answer as Alice based on the `channel_id`, and forwards the answer to Alice (6). After receiving the answer, a direct connection between Alice and Bob has been established (7).

the chunk start with a keyframe, hence the chunk contains enough information to be played independent of other chunks.

However, chunks can be large binary objects causing users with limited upload bandwidths to take a long time transferring individual chunks. The downloading user can have excess upload capacity, but have to wait for the whole chunk to be received before having anything to upload. Thus the upload capacity of the downloading user is wasted [SHM07]. For this reason, the chunks are divided into smaller units, called *pieces*, allowing users to forward individual pieces as soon as they are received, rather than having to wait for a whole chunk.

To be able to assemble a chunk from a collection of pieces, each piece header contains a chunk identifier (`chunk_num`) and a piece identifier (`piece_num`), indicating the position of the data in the original media stream. The piece identifiers are integers `piece_num` $\in [0, 255]$, hence they can be represented with an 8-bit unsigned integer. Chunk identifiers are represented by integers `chunk_num` $\in [0, 2^{16} - 1]$, which can be represented with a 16-bit unsigned integer. Additionally, a third field, `piece_count` represent the number of pieces that make up the complete chunk as an 8-bit unsigned integer.

| chunk_num | piece_num | piece_count | reserved | payload |
|---|---|---|---|---|
| 2 bytes | 1 byte | 1 byte | 4 bytes | $\leq 2^{16} - 8$ bytes |

Figure 4.2: The Protocol Data Unit (PDU) structure used when sending media data. The length of the header is 8 bytes and the maximum PDU length is $2^{16} - 1$ bytes.

In total the fields `chunk_num`, `piece_num` and `piece_count` can be represented in 32-bits. The size of the header used for the media datagrams is set to 64 bit, including bytes 4 through 7 to be reserved for implementation specific purposes. The structure of the PDU can be seen in Figure 4.2

### 4.2.3 Signaling

Signaling messages are exchanged within the overlay network as strings with a JSON serialized payload. The payloads are prefixed with a single character using the function given in Listing 4.1 to indicate the message type.

```
function createSignal(prefix, payload) {
    return prefix.charAt(0) + JSON.stringify(payload);
}
```

Listing 4.1: In signaling messages, the payload is prefixed with a single character denoting the message type.

## 4.3 Buffering and Playback

A peer must buffer received chunks within a sliding window. The window contains up to a few minutes worth of chunks, and individual pieces of the contained chunks are available to be sent to peers if requested. Hence, the sliding window contains chunks that have already been played, but also chunks scheduled to be played. Availability information used for requesting chunks from peers can be derived from the contents of the sliding window as described in [HLL+07].

With the correct encoding and a compatible browser, chunks of video can be buffered and played in a Hypertext Markup Language (HTML) video element using JavaScript APIs described in the W3C Media Source Extensions candidate recommendation [CBW15].

## 4.4 Overlay Network Topology

This section describes the Dugnad overlay network.

### 4.4.1 Virtual Coordinates

What characterizes the overlay network of Dugnad is the assignment of virtual coordinates to nodes in a n-dimensional vector space. These coordinates are not required to be bound to properties of a node, as the mere existence of the coordinate introduce some properties in the overlay network, as seen later in this section. In this thesis we look at an implementation where $n = 2$.

Introducing node coordinates is motivated by the lack of an obvious global node identifier in the WebRTC domain. In Pastry, numeric node identifiers are used to route a message to a target by selecting peers with numerically closer identifiers. In Dugnad, the same property is achieved by comparing Euclidean distances between nodes based on their virtual coordinates.

**Definition 4.1.** The distance between node $A$ and node $B$ is the Euclidean distance between the two nodes and is denoted $\delta_{A,B}$.

However, in Dugnad the goal is not to find one specific recipient of a message, but to broadcast data from a single source. Hence media is forwarded to peers for which distance to the broadcaster are greater than that of the node, thus the coordinate of the broadcaster has to be known to all nodes.

**Definition 4.2.** The distance between nodes $A$ and the broadcaster is $\mathcal{D}_A$

**Definition 4.3.** Node $A$ is considered a child of node $B$ if $\mathcal{D}_A > \mathcal{D}_B$. Otherwise, if $\mathcal{D}_A < \mathcal{D}_B$ $A$, then $A$ is a parent of $B$.

Given that nodes only forward data to peers who are children, the data-flows between peers are directed; hence the data plan becomes a DAG. Nodes with equal $\mathcal{D}$ can occur, thus it has to be considered when implementing Dugnad. In Figure 4.3 we can see an example of a graph with this property where the broadcaster is located in the origin.

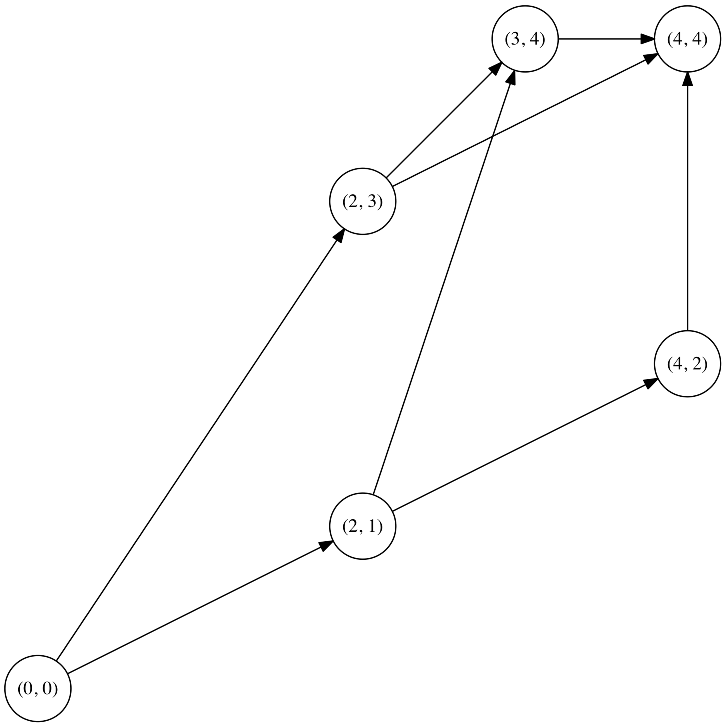


Figure 4.3: Nodes are assigned coordinates, and directions of edges are decided by node's Euclidean distance to the origin. This results in a Directed Acyclic Graph (DAG).

The broadcaster's coordinate must be known, and be the root of a topological ordering of the nodes. Since there is only one broadcaster in the overlay network, the origin, i.e. the coordinate zero, is reserved for the broadcaster. For all nodes other than the broadcaster, $\mathcal{D} > 0$, hence it is impossible to be a parent of the broadcaster.

In Figure 4.4, the nodes in Figure 4.3 are given in a topological ordering with the broadcaster node as the root.

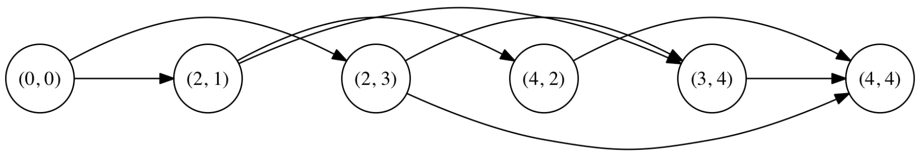


Figure 4.4: The nodes are displayed in a topological ordering based on their distance to the broadcaster. Directed edges suggest data-flows away from the origin.

For a message from the broadcaster to be able to reach all nodes in the directed graph, all nodes must be ancestors of the broadcaster. An implication of this is that the node with the lowest $\mathcal{D}$ at all times needs to be a direct child of the broadcaster. Additionally, all nodes except the broadcaster are required to have at least one parent.

Through using the implicit relationships created by the distance property, cyclic data-flows are eliminated in the overlay network for strict push-based behavior. However, this property alone is not a guarantee that a message from the broadcaster will reach all nodes. Additionally, there are potentially redundant paths in the overlay network, as can be seen in Figure 4.4, hence bandwidth waste can still occur in a push based scheme.

### 4.4.2 Coordinate Selection

Arriving nodes cannot be trusted to select their own coordinates. From Figure 4.4 we see that the node with the lowest $\mathcal{D}$ needs to be a direct child of the broadcaster node. Hence nodes could take advantage of this and always choose a coordinate giving them a small $\mathcal{D}$, hence forcing a direct connection to the broadcaster. This would result in data being received in a single hop for the node, as the broadcaster prioritizes it; if nodes unwilling to forward data claimed all direct connections to the broadcaster, the remaining nodes would not receive any data.

For this reason, the coordinates are chosen by the RP in the bootstrapping phase. This allows the server to keep track of which coordinates have been assigned, thus knowing which nodes are closest to the broadcaster. How the coordinates are chosen is not dictated by Dugnad, but is left to the implementation to decide. However, which nodes are given coordinates close to the broadcaster can impact the performance of the overlay network. Ideally, implementations should provide these coordinates to nodes who are offering high upload bandwidths.

## 4.5 Connection Management

**Viewer** Connections are established between nodes assisted by the RP for signaling purposes. An initial list of peers is suggested by the RP in the bootstrap procedure, and the viewer will initiate connections such that it has $n$ children and $n$ parents is possible. If a node's peer set contains fewer nodes than is desired, it can request more peers from the RP. All incoming connection establishment offers are accepted.

**Broadcaster** The broadcaster should keep $n$ connections in his peer set, which should contain the $n$ peers in the overlay network with the smallest $\mathcal{D}$. Arriving viewers with small $\mathcal{D}$ will find the broadcaster in the proposed peers during the bootstrap procedure, hence they will be able to establish connections to the broadcaster.

A broadcaster with fanout $k$ will send $k \leq n$ copies of the stream to the $k$ connections in its peer set with the lowest $\mathcal{D}$.

## 4.6 Peer Selection

Data flow in the DAG should be influenced by the topological order given in Figure 4.4 to avoid cyclic data-flows, meaning that data should flow from the broadcaster to nodes with increasing $\mathcal{D}$, i.e. flow from a node to its children.

### 4.6.1 Gossiping: Push

Whenever a node receives a new useful piece it should push that piece to one of its children. The receiving child is selected based on the distance between the node and the child, and children closer to the node are more likely to be selected. This can be described by looking at Table 4.2. The table incorporates the nodes from Figure 4.3, but assumes a complete graph, hence there is an edge between all nodes. When the node at coordinate $(2, 1)$ receives a new useful piece, it will choose a child for which to forward the piece. The children are ordered ascending on distance $\delta$, hence the node at $(2, 3)$ is most likely to be selected, followed by $(4, 2)$. The least likely receiver is $(4, 4)$. Note that in the table, all of the children have a $\mathcal{D}$ greater than that of the node. However, the ordering of nodes according to $\mathcal{D}$ and $\delta$ is not necessarily the same: A node with coordinates $(1, 3)$ would be the child with the smallest $\mathcal{D}$ as $\mathcal{D} = \sqrt{10}$, but the distance between the nodes $\delta_{(2,1),(1,3)} = 5$ means that it would not be the closest peer.

The purpose of the push mechanism is to quickly spread rare pieces so that they are made available for dissemination through pulling. Allowing parents to push new data units to their children helps to rapidly diffuse new data units, reducing the delay for nodes that are several overlay network hops away from the broadcaster [MR06]. Never pushing to a parent eliminates the possibility of creating push cycles as discussed in subsection 3.4.1. However, if both the node at $(2, 1)$ and $(2, 3)$ are direct children of the broadcaster, the node at $(2, 1)$ would push redundantly with a high probability, hence an implementation should include mechanisms to detect and close such connections.

This can be described as a weighted random peer selection process. The weight function used to select the peer is a function of the distance between a node and its neighbor [MP10].

### 4.6.2 Gossiping: Pull

The pieces a node does not receive through pushing are obtained through pulling from other nodes. This is done through sharing availability information with peers,

| | Coordinate | $\mathcal{D}$ | $\delta_{(2,1),(x,y)}$ |
|---|---|---|---|
| Parents | $(0,0)$ | $0$ | $\sqrt{5}$ |
| - | $(2,1)$ | $\sqrt{5}$ | $0$ |
| Children | $(2,3)$ | $\sqrt{13}$ | $\sqrt{4}$ |
| | $(4,2)$ | $\sqrt{20}$ | $\sqrt{5}$ |
| | $(3,4)$ | $\sqrt{25}$ | $\sqrt{10}$ |
| | $(4,4)$ | $\sqrt{32}$ | $\sqrt{13}$ |

Table 4.2: Distances to the peers of a node with the virtual coordinate $(2,1)$.

implicitly asking the peer to push unavailable pieces that he possesses.

The peer to send a pull request to is chosen similarly to a push target. However, both parents and children are allowed to receive pull requests. When selecting a peer, the node order all parents and children ascending according to their distance $\delta$ to the node as is done in Table 4.2; the ordered list of children is appended to the ordered list of parents making a list of peers $L$. $L$ has the property that all parents have indexes lower than the children in the concatenated list. When selecting a peer, an index in $L$ is selected, where the probability of selecting an index $i$ is inversely proportional to $i$ where $i = 0$ yields the highest probability. Hence there is high probability of selecting a parent with a small distance, and unlikely to select a child.

Due to the possibility that nodes can be orphaned, i.e. loose all its parents due to churn, they have to be able to pull from their children to be able to receive pieces during the time they try to locate new parents. Thus the non-cyclic data-flow property does not apply for pulled data.

## 4.7 Implications of Virtual Coordinates

The argument given in subsection 4.4.2 shows that the coordinates must be issued from a central authority, i.e. the RP. This has some interesting implications: in Gridmedia [ZLZ+05], the RP suggest peer candidates that have long IP-address prefix matches with the node to connect nodes behind the same NATs and subnets. Grouping nodes with close IP-addresses in Dugnad, such that they are close in proximity in the coordinate system serves the same purpose, and is feasible since the RP in Dugnad is also responsible for suggesting the initial peer candidates.

The IP address space is one-dimensional, and proximity in this regard is not always a good estimator of network latency. Virtual coordinates in Dugnad are not limited in number of dimensions, allowing properties such as geographical location and measurements of Round-Trip Times (RTTs) to be introduced as separate dimensions in the coordinates. This way coordinates can carry information relating to properties of a node, as semantics are added to the coordinate dimensions.

Allowing individual nodes to manipulate some dimensions of their peers' coordinates, such as adaptively updating an estimated value for EED, can be done to improve peer selection through favoring nodes with short network distances. Using the same idea, a node can effectively choke a peer, i.e. ask it to stop sending data, by requesting the peer to temporarily change the node's coordinate such that the node ends up with a distance further away from the peer. Note that being able to instruct peers to change your coordinate such that the $\mathcal{D}$ increases does not enable a node to take advantage in the way discussed in section 4.8.

## 4.8 Caveats

The RP needs to maintain a WebSocket connection to the nodes in the overlay network in order to be able to assist in signaling, where in traditional P2P systems the RP only needs to store the IP address and port number.

Coordinates require some static dimensions that can easily be derived by the RP. Ideally, benign nodes with high forwarding capacities should be awarded coordinates close to the origin, but these properties are not possible to see in a node a priori. Hence a distributed algorithm to allow identification of high capacity nodes should be included in an implementation, such that these nodes can be promoted and change their coordinates to better exploit their capacity. Adding semantics to the coordinate dimensions, like relating them to properties such as RTTs or IP address prefixes comes with an increased complexity in the coordinate assignment process of the RP.

The amount of connection establishment signaling that has to be handled by the RP is growing with the number of nodes in the overlay network, despite this being undesired as listed in section 3.6. This can be changed by allowing nodes in the overlay network to assist in establishing connections between their peers without the involvement of the RP. However, this has a downside in the absence of a trusted party conveying signaling messages, which imply that coordinates can no longer be trusted. Two malicious nodes can establish connections to a benign third party deceived to believe they have coordinates where $\mathcal{D}$ is close to 0. This can ultimately lead to a scenario where all the peers of the broadcaster are malicious nodes, and thus service can be denied for all remaining nodes.

Note that a similar attack is still possible if the coordinate given upon bootstrap is random. A malicious node can simply rejoin until he receives a coordinate in the desired range. Hence the coordinates should not be purely random.

## 4.9 Dependability

There are concerns that must be addressed in an implementation of Dugnad concerning dependability of the system, as a range of faults can happen [ALRL04].

With regard to the availability of the service, RP is a single point of failure, as it is the only entry point to the overlay network. Due to the WebSocket connections, the nodes in the network will be able to contact the RP after it recovers. All nodes should not do this immediately as this could cause the RP to go back down if the overlay network consisted of a big number of nodes.

The information associating nodes to broadcast channels and coordinates can be lost if the RP fails. However, if the important information such nodes coordinates are cryptographically signed by the RP, the information can be retrieved upon recovery. Additionally, the RP can trust the broadcaster; hence partial recovery should be possible using strictly the information provided by the broadcaster. Details regarding recovery of the overlay network have not been studied as part of this thesis.

From a reliability perspective, the existing connections in the overlay network will not be affected if the RP fails, as they will continue to receive service from the broadcaster. However, establishment of new connections between nodes in the existing overlay is not possible without the RP as long as nodes are not trusted to assist in connection establishment.

Maintaining the integrity of the system is a challenge, since the nodes in the overlay network are running JavaScript application that can easily be modified by each client. Hence the risk that nodes may misbehave is prevalent.

Chapter 5

# Experimental Implementation

In the following chapter, an experimental implementation of Dugnad is presented.[1]

## 5.1 Technology Stack

The viewer application is designed to work natively in a web browser with a minimum of dependencies. The broadcaster application is a modified viewer application, also running in the browser. The RP is an application built on the server side event-oriented platform Node.js [ALMC14] and relies on the Socket.IO library[2] for handling WebSocket connections. The RP serves the viewer and broadcaster applications over HTTP. Due to WebRTC still being an experimental technology, the implementation is only tested in the Google Chrome and Chromium Web Browsers.

## 5.2 Data Model

Following is a description of how some the concepts from Dugnad are represented and used in the implementation of the RP.

### 5.2.1 Broadcast Channels

For simplicity, the experimental implementation only supports a default broadcast channel, hence multiple broadcasters cannot simultaneously use the same RP.

In the RP, a broadcast channel contains all the sockets connecting the RP to nodes in the system as well as their virtual coordinate. Hence, for a given broadcast channel and an incoming message on a socket, the RP can know the virtual coordinate of the node originating the message, and also find sockets to nodes who are parents and children of the node.

[1]The source code can be found at https://github.com/martme/dugnad
[2]The Socket.IO library is available at http://socket.io/ (Accessed 27. May 2015)

Signaling channels represent a map between a unique identifier (`channel_id`) and two sockets, allowing the RP to forward a series of messages back and forth between two communicating parties. A signaling channel is only open for a limited time (8 seconds in the current implementation), thus nodes have a limited window of time for which to establish connections. The limited lifetime of the channels is chosen to reduce the resource requirements for storing them in memory.

### 5.2.2 Node Identifiers

A node is uniquely identified by the RP using the `id` of the Socket.IO socket for the WebSocket connection from the RP the node. Nodes uniquely identify their peers using the `channel_id` of the signaling channel used to establish a connection to that peer. Hence a node can potentially establish multiple connections to the same node, as a new signaling channel will have a different `channel_id`. For this reason, a node will never establish a connection to two nodes with the same coordinate.

### 5.2.3 Coordinate Selection

The coordinate assignment policy of the RP is uniformly random, providing two-dimensional coordinates using the code given in Listing 5.1. Coordinates carrying semantics, as briefly discussed in section 4.7 are not incorporated in the current implementation.

```
var coordinate = {
    "x": Math.random()*99 + 1,
    "y": Math.random()*99 + 1
}
```

Listing 5.1: Assignment of coordinates to arriving viewers is done using the following JavaScript procedure.

## 5.3 Signaling Interfaces

In Socket.IO, messages sent between a client and the web server (viewer/broadcaster and the RP) consist of an event name and a message. The event name is used as a type field in the implementation; hence different types of signaling messages are distinguished by the named event triggered when receiving the message. The following subsection describe the different signaling interfaces, i.e. how the signal message types are interpreted and acted upon by the RP.

### 5.3.1 Create

The broadcaster sends type `create` signal to the RP in order to initiate a stream. Contained in the payload is the broadcast channel identifier belonging to the broadcaster. The RP replies with is an empty message of type `created` if the request was successful.

A successful request of this type causes the RP to initialize a broadcast channel where the broadcaster is the only node. If a node initiates a broadcast channel using this interface, it will be given a coordinate in the origin of the virtual vector space, hence be assigned the broadcaster role. This is the only time this coordinate is assigned, and the only way to initialize a broadcast channel.

Restrictions to who may initialize a broadcast channel using a given broadcast channel identifier can be enforced by the RP. However, this is omitted in the experimental implementation to reduce complexity.

### 5.3.2 Join

Viewers must send a request to the RP in order to join a broadcast channel. The request is a signaling message with type `join`, where the payload contains the room identifier, i.e. the reference to which broadcast channel the viewer wants to join. The RP responds to the signal with a `joined` message acknowledging the request to join was successful.

The current implementation does not use information such as node capabilities contained in the payload, as mentioned in subsection 4.1.2.

### 5.3.3 Bootstrap

A successful `join` request is followed by a `bootstrap` request. The bootstrap procedure is depicted in Figure 4.1 and Table 4.1. Upon receiving a signal with type `bootstrap` from a viewer, the RP will assign a coordinate to the node and associate it to the socket in the broadcast channel.

The RP responds to the bootstrap request with a `bootstrap_channels` signal that contains several `channel_ids` and the associated coordinate of the other party of that signaling channel.

### 5.3.4 Patch

A node can send `patch` signal to request more signaling channels from the RP. The response is a list of identifiers of signaling channels like in the bootstrap interface described in subsection 5.3.3. The purpose of this interface is to allow nodes to establish new connections when churn causes peer sets to shrink, or a node has fewer peers than desired. In contrast to the bootstrap interface, which can only be used once upon joining the overlay network, a node is allowed to send multiple patch requests.

If a patch request originates from a node that has less than 4 logical parents among the nodes in the broadcast channel, a signaling channel between the node and the broadcaster is included in the returned channel list.

### 5.3.5 Signaling

The JSEP signaling required for establishment of P2P connections in WebRTC is sent using the `signaling` event. All messages sent using the signaling interface must contain a `_channel_id` in the JSON payload indicating which signaling channel the sender wants to use. Three types of messages are sent; offers, answers and Interactive Connectivity Establishment (ICE) candidates, and are identified by the `_type` value contained in the payload. Messages where `_type` is set to `offer` must be intercepted by the RP such that the coordinate of the originator node is included in the connection establishment offer. The implementation of the forwarding function for JSEP messages including the intercept mechanism is given in Listing 5.2.

```
function forward (socket, message) {
    var obj = JSON.parse(message);
    var channel_id = obj._channel_id;
    var receiver = getReceiverInChannel(channel_id, socket);
    var sender = getSenderInChannel(channel_id, socket);
    if (receiver && sender) {
        if (obj._type === "offer") {
            obj._coordinate = sender.coordinate;
        }
        receiver.socket.emit(JSON.stringify(obj));
    }
}
```

Listing 5.2: Forwarding session negotiation messages between parties of a signaling channel. The socket parameter is the Socket.IO socket between the Rendezvous Point (RP) and the originator of the message.

### 5.3.6 Disconnect

A viewer disconnecting from the server will send a `disconnect` signal. Upon receiving a disconnect, the server removes the reference to the viewer from the broadcast channel, hence active signaling channels referencing the disconnected node will no longer work.

## 5.4 Dugnad Node Design

A system diagram of a viewer node in Dugnad is given in Figure 5.1. The system design is inspired by the design of nodes in DONet [ZLLY05].

The Connection Factory is responsible for handling signaling with the RP; when a connection has been established, it is sent to the Connection Store. Connections are `PeerConnection` objects with an open `DataChannel`, allowing nodes to communicate directly with other nodes in a P2P manner.

The Connection Manager maintains information about the connections in the Connection Store, and has interfaces to push and pull data to peers. It is responsible for sorting the connections according to their coordinates, and distinguishing between parents and children.

When data is received on a connection, it is sent to the Assembler and registered in the Buffer Map. When all pieces of a chunk is received, it is assembled and made available to the media player.

The Buffer Map is an index of all chunks known to the viewer. When the assembler receives a piece, it notifies the Buffer Map; if the piece qualifies to be pushed, it is sent to the Scheduler which finds an appropriate target connection using the interfaces of the Connection Manager. Additionally the Scheduler is responsible for deciding which data, if any, to send as a response to pull requests, and schedule the response.

## 5.5 Connection Management

### 5.5.1 Timer Based Connection Management

Since implementation uses UDP as the underlying protocol for data channels, the underlying protocol does not inform the application layer when the remote party closes a connection. Timer based connection management is applied in the implementation, where connections expire if a party is idle for 10 seconds. To keep connections from expiring when there is no data to send, heartbeat messages are sent every 3 seconds if no other data has been sent. These messages are overlay network signaling messages represented with a single character prefix `h` and an empty payload. The heartbeat messages are encoded according to the procedure given in Listing 4.1.

### 5.5.2 Choosing the Size of Peer Sets

In [MR06], it is suggested that to minimize the required buffer interval in swarming based P2P live streaming system, nodes should have a peer-degree no higher than 16 when peers have 1.5 Mbps upload bandwidth. Based on these numbers, the value $n = 7$ is used in the implementation for the desired number of children and parents, hence the desired total number of peers becomes 14. However, the ideal value depends on an individual peer's bandwidth. Exploring the implication of this parameter choice should be subject to future work.

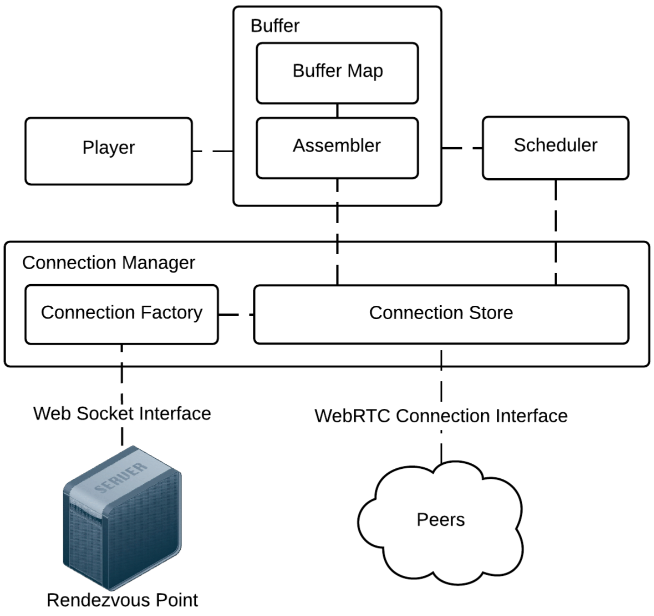


Figure 5.1: Client architecture overview.

### 5.5.3 Defeating Wasteful Connections

Logical shortcuts can occur in the implicit directed graph dictated by nodes' virtual coordinates. This can cause waste of bandwidth due to nodes blindly pushing data to children who already possess the data as pointed out in subsection 4.6.1. A mechanism seeking to address this issue has been implemented and is based on scoring individual connections based on the data they carry.

All new connections start with the score 4 to give a head start to new connections. The score is incremented by 1 for every piece transmitted and received, and for each pull request sent, on a connection. After a piece has been received and the score has been incremented, the piece header is inspected. If the node already possessed the piece, the transmission was redundant. As a result, the new connection score $S$ is decreased according the formula $S = max(1, S/2)$

Every 10 seconds, the scores for all connections are tallied, and a node with excess connections, that is more than 7 parents or children, drop the single lowest scoring connection in that group. Hence, if a node has 9 parents and 5 children, the node would drop the lowest scoring parent, reducing the count to 8, while keeping all 5 children. A node with 9 parents and 10 children will drop one parent and one child.

Note that the score is computed locally, and the score of a connection will differ between the two parties. Say that Alice's highest-ranking child is Bob, but Bob receives mostly redundant pieces from Alice due to a shortcut in the overlay network. Bob will quickly give Alice a score close to 1, thus it is likely that Alice becomes Bob's lowest ranking parent. Thus Bob will drop the wasteful connection.

### 5.5.4 Maintaining a Healthy Connection Count

A node may have fewer than the desired number of 7 parents or children. Every 30 seconds, a node checks if this is the case, and if so, sends a patch request to the RP.

Viewers who receive connection establishment offers accept them if their total number of connections for that group is less than two times the desired number. Thus if an incoming offer originates from a child, the connection is only accepted if the node's number of children is less than 14. However, if the node already have a peer at the coordinate which is contained in the offer, the offer is rejected.

## 5.6 Peer Selection

A viewer selects peers for push and pull according to the procedure given in section 4.6. A peer is selected from the ordered list of peers produced by the given procedure by selecting an index in the list. The peer chosen is the peer corresponding to the selected index. Selecting the index is done using a weighted function such that the returned value is in the range $i \in [0, l-1]$ where $l$ is the number of peers to select from. The probability of selecting a given index is a function of the index, where 0 grants the highest probability, and the procedure used for selecting the index is based on a geometric distribution.

## 5.7 Chunk Buffer

In the implementation, the Buffer is used to store chunks that have been successfully assembled. The lowest numbered chunk that has not yet been assembled is called the *buffer head*, and the initial value of the buffer head is set after receiving three consecutive chunks. The initial value is the `chunk_num` of following the highest numbered chunk out of the three.

If two consecutive chunks are received that both have `chunk_num` greater than the buffer head, the buffer head is moved past these chunks, hence the viewer will give up on the incomplete chunk.

## 5.8 Chunk Selection

### 5.8.1 Disseminating Rare Pieces: Push Gossiping

In the description of the Dugnad push mechanism given in subsection 4.6.1 it is stated that nodes should push when receiving a *new useful piece*. In the current implementation a piece is only pushed if it is the piece with the highest chunk and piece number received by the node. This means that if pieces are received out of order, only the highest numbered pieces are pushed. A motivation for this limited push behavior is the lack of a limited time interval where a segment of a given chunk can be pushed, which is a mechanism described in [MR06]. This is omitted in the implementation to reduce complexity of the client. Also, less conservative push policies have caused a lot of wasteful pushes to the nodes furthest away from the broadcaster in preliminary testing. Thus, a more efficient push mechanism should be introduced in a later version.

### 5.8.2 Exchange of Availability Information: Pull Gossiping

A single pull request is sent to a selected peer every 500 milliseconds. The requests are signaling messages prefixed by the character `r` as defined in subsection 4.2.3.

The pull request payload consists of an integer `chunk_num`, and a list `pieces` of integers representing piece identifiers; the tuple `(chunk_num, piece_num)` represents the chunk and piece identifiers of a chunk not possessed by the sender. All pieces missing to complete the given chunk is listed in the request. An example payload is given in Listing 5.3.

```
{
    "chunk_num": 5,
    "pieces": [1, 3, 6]
}
```

Listing 5.3: A JSON formatted pull request payload. In this example, the peer is requesting pieces 1, 3 and 6 from the chunk with identifier 5.

The `chunk_num` requested is the current value of the buffer head. Before the viewer has initialized the buffer head, the requested `chunk_num` is the highest `chunk_num` seen on any piece received up until that point. Before a viewer has received any pieces, it will send a wildcard pull request. In this request the `chunk_num` is set to

`*` and the `pieces` list only contain the number 0. When responding to a wildcard pull request, a peer will push the newest piece he possesses, i.e. the piece with the highest `chunk_num` and `piece_num` tuple.

When receiving a regular pull request, a viewer checks the availability of the requested pieces. All available pieces are pushed to the originator of the request.

In the current implementation, the availability information exchanged is not utilized in the peer selection step, as the selection is only weighted on distances in the virtual coordinate space. For a more effective push based diffusion scheme, a mechanism such as that of CoolStreaming/DONet could be applied [ZLLY05]. This should be possible within the confinements of the Dugnad architecture, as the Buffer Map and general architecture is inspired by that of CoolStreaming.

## 5.9 Broadcaster Peer Selection

The broadcaster node splits chunks of media content into pieces. Each piece is forwarded up to $k$ times where $k$ denotes the fanout. If the broadcaster has $k$ or more peers, each piece is forwarded exactly $k$ times. The peers are sorted according to their distance to the broadcaster in the virtual coordinate space, and the $k$ closest peers are chosen. Broadcasters should be able to select the value $k$ individually depending on their available bandwidth, but in the current implementation $k$ is set to 3.

Chapter 6

# Performance Measurements

The following chapter describe the experiments and results obtained using the implementation given in chapter 5.

## 6.1 Experiment Setup

### 6.1.1 Data Collection

In the experiments, data was gathered by capturing events at the nodes in the overlay network. These events were sent to the RP and recorded together with events captured by the RP. The following events were recorded:

**Join** A node joins the overlay network. This event is recorded at the RP when a bootstrap request is received.

**Leave** A node leaves the overlay network. This event is recorded at the RP when the WebSocket connection to a node is disconnected.

**Spawn** The broadcaster has spawned a new chunk that is ready to be forwarded to the overlay network.

**Receive** A node has successfully received and assembled all pieces of a chunk.

The time associated to an event is the time at which it was received at the RP. Hence for events captured by nodes in the overlay network, the time includes the EED between the node and the RP.

### 6.1.2 Practical Limitations

All the nodes were run in computers residing in the same local network providing high-speed campus access. According to Table 6.1, the global average upload bandwidth for mobile connections in May 2015 was 5.0 Mbps; average download bandwidth was 12.3 Mbps. This suggests that the high bandwidth in the local network can yield results that are in accordance with what could have been observed using nodes spread around the Internet, if the bandwidth load on the individual nodes is well within these values. However, the results have to account for the very low RTTs between nodes.

| Type | Download | Upload |
|---|---|---|
| Broadband | 23.2 Mbps | 10.6 Mbps |
| Mobile | 12.3 Mbps | 5.0 Mbps |

Table 6.1: Global average measured bandwidth for broadband and mobile connections collected from http://www.netindex.com/ (Accessed 12 May 2015).

Since the streaming application is designed for use within a web browser, all nodes were run in graphical web browsers during the experiments. Due to a limited number of computers, several viewers were run on the same host computer. The experimental setup is illustrated in Figure 6.1; to run a single experiment, all 18 computers must be manually instructed; hence each experiment is very time consuming.

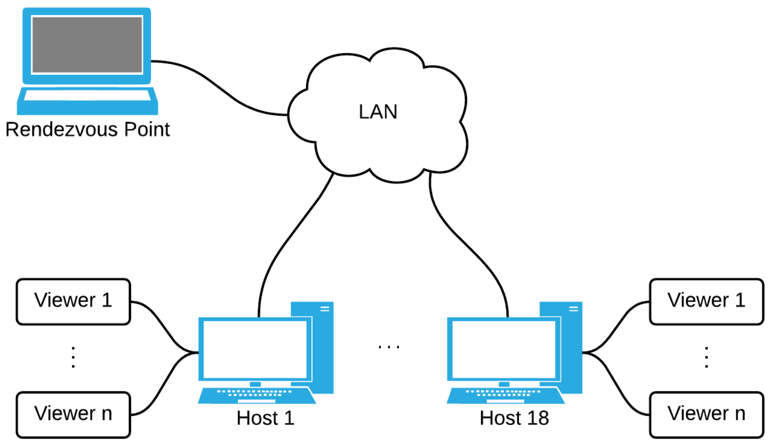


Figure 6.1: A total of 18 host computers and a laptop were used during the experiment. Each of the hosts ran multiple viewer instances.

Further increasing the number of nodes from a limited pool of host computers was done using a wrapper application for the viewer program. Multiple instances of the viewer web application were loaded into separate `<iframe>` elements, hence started and stopped without human interference. This application ran on all the host computers during the experiments, and the RP was hosted on a separate laptop.

### 6.1.3 Video Stream Generation

In all the experiments, a bit stream with a constant bit rate $b = 2500$ Kbps split into chunks representing 3 seconds of data is generated in the broadcaster and served to the overlay network.[1] These chunks are again split into pieces with a maximum size of 16 KB, the same size as data units in the BitTorrent protocol [Coh08]. The fanout of the broadcaster $k = 3$ and the upload capacity of the broadcaster is sufficient for serving all required copies of the stream. Looking at Table 6.1, we see that the required upload bandwidth for the media stream, 7500 Kbps for all three copies, is less than the global average upload speed for a broadband connection.

## 6.2 Experiments

The implementation of the application providing multiple viewers per host computer is given in Listing 6.1. Parameters for node arrivals and churn are selected based on the following constraints.

1. Users arrive according to a Poisson process with rate $\lambda$.
2. The durations users stay in the system is exponentially distributed with the parameter $\mu$.
3. A single computer should with a high probability host no more than $c$ users at any given time.

### 6.2.1 Experiment A: With Churn

Finding parameters that prevent computers from hosting more than $c$ nodes is achieved by calculating the probability of blocking in an M/M/$c$ queuing system using the Erlang's C formula [Ive10]. Choosing the number of servers $c = 8$, and service time $\mu = 1/4$, i.e. expected lifetime of 4 minutes, we set the probability of a user being served instantly upon arrival to 90%, i.e. the probability that an arriving node would have to wait for service in the M/M/8 model is $C(c, \lambda/\mu) = C(8, \lambda/\frac{1}{4}) = 0.1$. Solving the Erlang's C formula for $\lambda$ with these parameters yields an arrival rate of $\lambda = 1.12$ users per minute.[2]

The actual node life cycle can be described as an M/M/$\infty$ queue since a nodes service time, i.e. lifetime, is independent of the number of other nodes hosted on the

[1] A bit rate of 2500 Kbps with a keyframe frequency not exceeding one per 4 seconds is recommended by YouTube for a 720p video stream, according to https://support.google.com/youtube/answer/2853702?hl=en (Accessed 12 May 2015).

[2] Solved using an online tool available at http://www.math.vu.nl/~koole/ccmath/ErlangC/index.php (Accessed 24. May 2015).

same computer. Thus, the expected number of nodes on a single host computer is given by $\lambda/\mu$ when $t \to \infty$.

### 6.2.2 Experiment B: Stable Overlay

The second experiment use the arrival rate $\lambda = 2.24$; the service time is infinite, i.e. $\mu = 0$ thus there are no node departures. To limit the number of clients hosted per computer in this experiment, arriving nodes were discarded if the system already held 6 nodes, hence all but the first 6 arriving nodes were discarded.

```
var EX_A = "A";
var EX_B = "B";
var EXPERIMENT = EX_A;

var container = document.getElementById("iframes");
var client_url = "/";
var lambda = 1.12 * (1/60);     // arrivals per minute
var mu = 1/4 * (1/60);          // lifetime

function exponentialDistributionSample(rate) {
    return -1 / rate * Math.log(Math.random());
}

function addUser() {
    var iframe = document.createElement("iframe");
    iframe.src = client_url;
    container.appendChild(iframe);

    // queue user removal
    if (EXPERIMENT === EX_A) {
        setTimeout(function () {
            container.removeChild(iframe);
        }, exponentialDistributionSample(mu) * 1000);
    }
}

function queueNextUserArrival() {
    setTimeout(function () {
        if (EXPERIMENT === EX_A ||
           (EXPERIMENT === EX_B && container.childNodes.length < 6))
                {
            addUser();
        }
        queueNextUserArrival();
    }, exponentialDistributionSample(lambda) * 1000);
}
queueNextUserArrival();
```

Listing 6.1: Running multiple viewers within a single browser window.

### 6.2.3 Limitations and Error Sources

Running multiple clients on the same host results in some of the P2P traffic being directed between nodes running on the same computer; hence no network traffic is generated since the data never leaves the host. However, this allows for a statistically sound modeling of churn and arrivals.

Bandwidth limitation of the nodes would be an interesting addition to the experiment setup, with a heterogeneous node population the nodes distributed according to the values given in Table 6.1. However, static host-level bandwidth limitations cannot easily be applied on hosts running multiple instances of the viewer web application since the observed limitations would depend on the number of instances sharing the network resource. Hence, experimentation using bandwidth-limited nodes is left for future work. The same goes for saturated network links and packet loss, which is not prevalent in the applied experimental setup.

The initial phase of each experiment involves manually starting the application given in Listing 6.1 on the host computers, hence some effects of startup delay is included in the results and the arrival rates are accelerated during the initial phase.

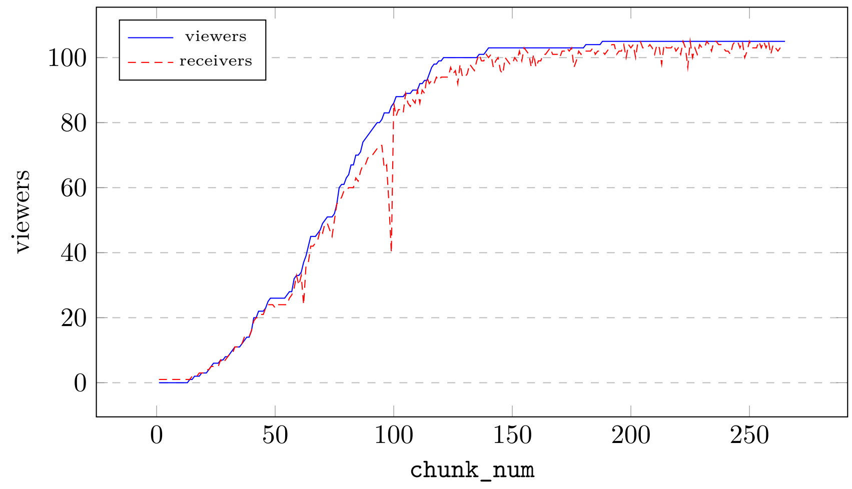
## 6.3 Results

This section contains results derived from the events recorded at the RP during the experiments.

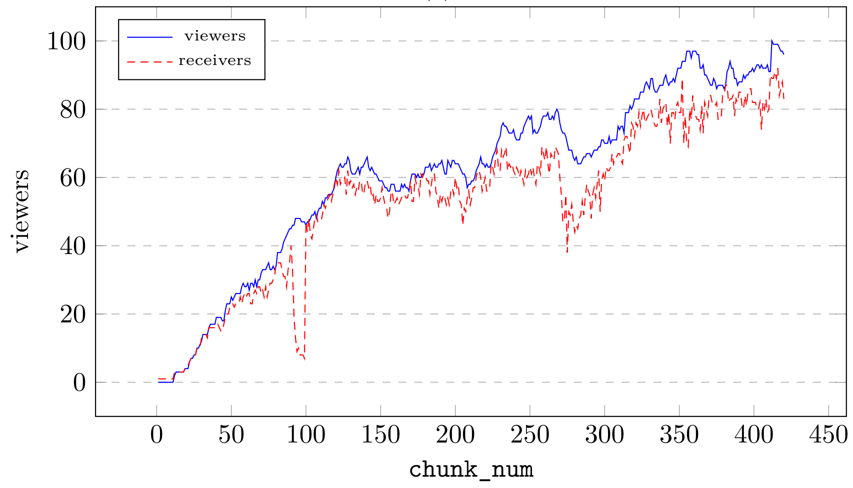
### 6.3.1 Viewer Numbers

The number of viewers in the overlay network over the duration of the experiments is given in Figure 6.2. In Figure 6.2a we see that for the experiment without churn, a stable number of nodes is reached around the spawn time of chunk 150, and the count remains stable beyond chunk 250; the presented results based on this data set is restricted to data for chunks in this interval. For the churn experiment, the interval used chunks 200 through 400. Note that in both cases the maximum number of nodes in the overlay network during the chosen interval is approximately 100, and that the time taken between each spawned chunk is 3 seconds.

Looking at Figure 6.2 we see that not all viewers receive every chunk, and that this is more prevalent in Figure 6.2b, i.e. under the effect of churn. However, in both cases, the majority of the nodes receive each chunk. This suggests that the remaining nodes should be able to receive their missing pieces given a more efficient availability-driven pull policy, as the pieces missing for one of these nodes to complete a chunk are likely to be available among its peers.



(a) Stable



(b) Churn

Figure 6.2: Number of viewers in the overlay network when the broadcaster spawns a given chunk compared to how many viewers receive a spawned chunk.

A possible explanation for the failed delivery of chunks in the stable environment is the implementation of the Buffer; as described in section 5.7 the buffer head can be incremented if a viewer receives two consecutive chunks with chunk identifiers higher than the current buffer head due to pieces being pushed from parents. In the experiment with churn, nodes may leave the system shortly after a chunk spawns and before they receive the chunk, hence they are only included in the viewer count for a given chunk. Additionally, chunks can contribute to the receiver count despite not being in the system when the chunk spawned.

The big negative spike in the number of receivers seen in both Figure 6.2a and Figure 6.2b around chunk 100 is likely caused by a change in the selected nodes to receive data directly from the broadcaster. In Figure 6.2b this is affecting multiple chunks, suggesting that some of the nodes being served by the broadcaster disconnected, thus causing a reduction to the broadcaster's effective fanout until the churn was detected by a connection timeout. A probable cause in the stable environment is a scenario where the broadcaster starts forwarding data to a node that has not yet established connections to a desired amount of children. In both cases, the children of the node previously receiving data directly from the broadcaster can become starved for a period of time, causing the buffer head to increase.

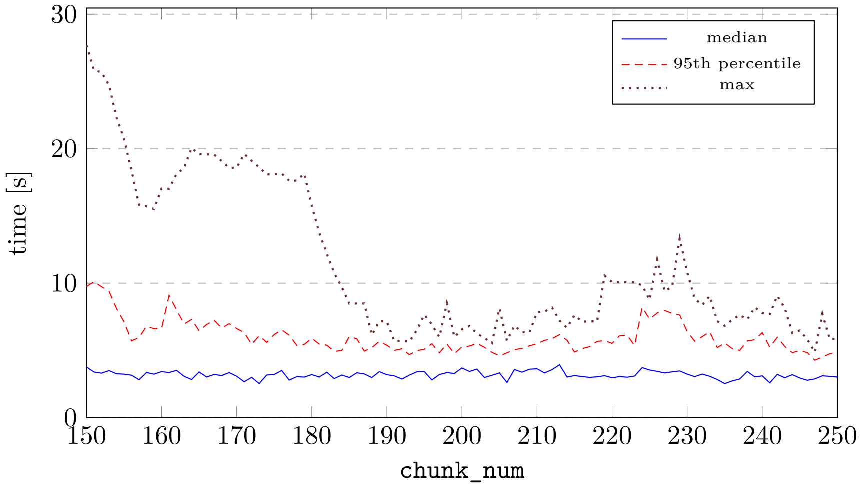
### 6.3.2 Diffusion Delay

Figure 6.3 shows the total *diffusion time* for each chunk in the selected intervals. The diffusion time denotes the time taken from a given chunk spawns at the broadcaster until it is assembled in nodes in the overlay network. However, since not all nodes receive each chunk, the total diffusion time denotes the time taken from the chunk originated until the last report of a node receiving the complete chunk reached the RP.

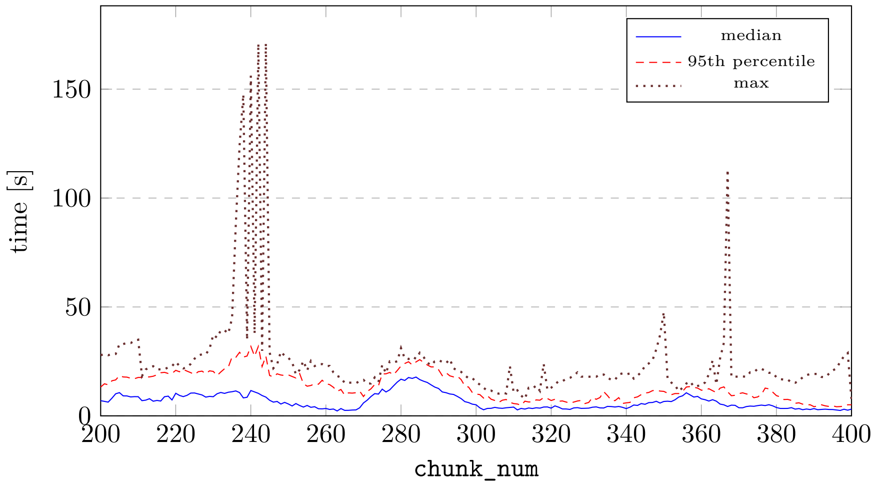
It is clear from Figure 6.3 that both with and without churn in the overlay network, the last 5% of the receiving nodes amounts to a substantial part of the total diffusion time of a chunk. Looking at Figure 6.3a we also see that if the overlay network has time to stabilize, the time taken to reach the last 5% of the nodes drop notably.

Due to the low RTTs between the computers used in the experiment, the observed diffusion time is lower than what would be observed if computers were not residing in the same campus network due to very low propagation delays and few network hops between nodes. Thus the number of hops taken in the overlay network by received pieces is an interesting property. This was measured by using one of header fields reserved for implementation purposes as depicted in Figure 4.2, to track the number of hops taken by each received piece. Each recorded receive event contains the single highest hop count found on in the pieces contained in the assembled chunk. The results are given Figure 6.4.

As indicated in Figure 6.4a all nodes receive chunks with an observed hop count of approximately 15 or less, with the average being less than 10. Given the fact that the broadcaster only supplies 3 copies of each chunk, these numbers show that the network is able to replicate the chunks. If viewers only forwarded a single copy per



(a) Stable



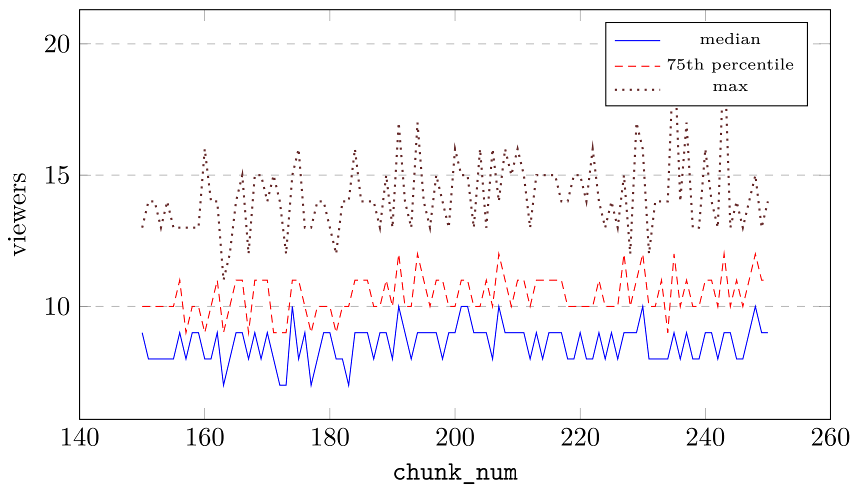
(b) Churn

Figure 6.3: Diffusion time of individual chunks.

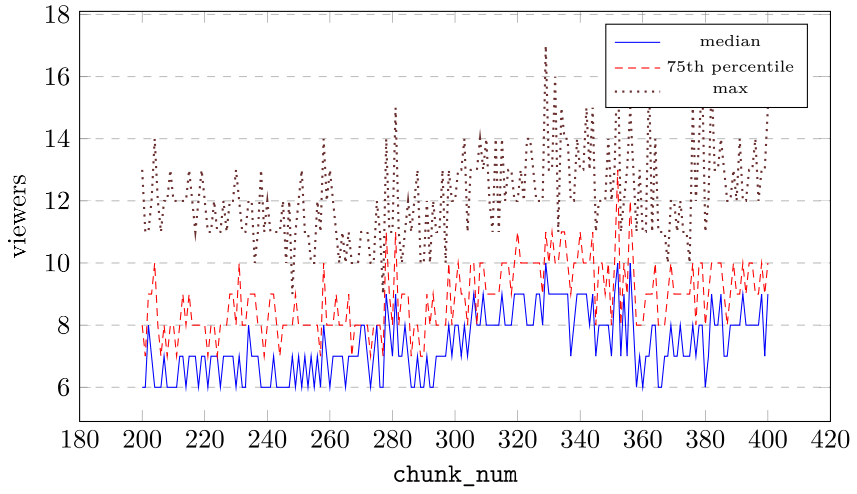
piece, the lowest possible number of hops would be $n/3$ for $n$ viewers organized in equally long node-disjoint linear chains, i.e. more than 30 for $n = 100$.

### 6.3.3 Bandwidth

The viewers measure the amount of data they have uploaded by tallying the number of bits sent to peers. Every 3 seconds, the node's records the number of bits sent during that period, and calculates a rolling estimator of the bandwidth usage $b_i$. The



(a) Stable



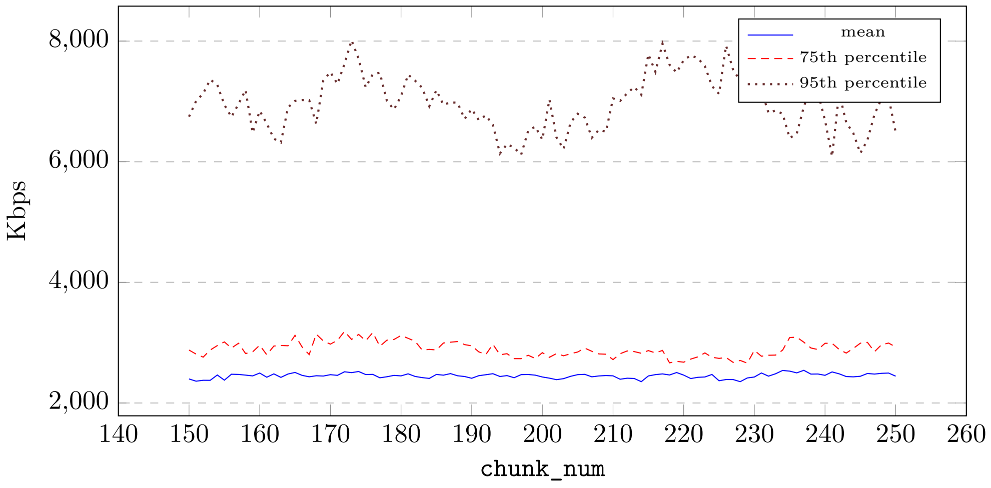
(b) Churn

Figure 6.4: Hop count for individual chunk based on max hop count for contained pieces.

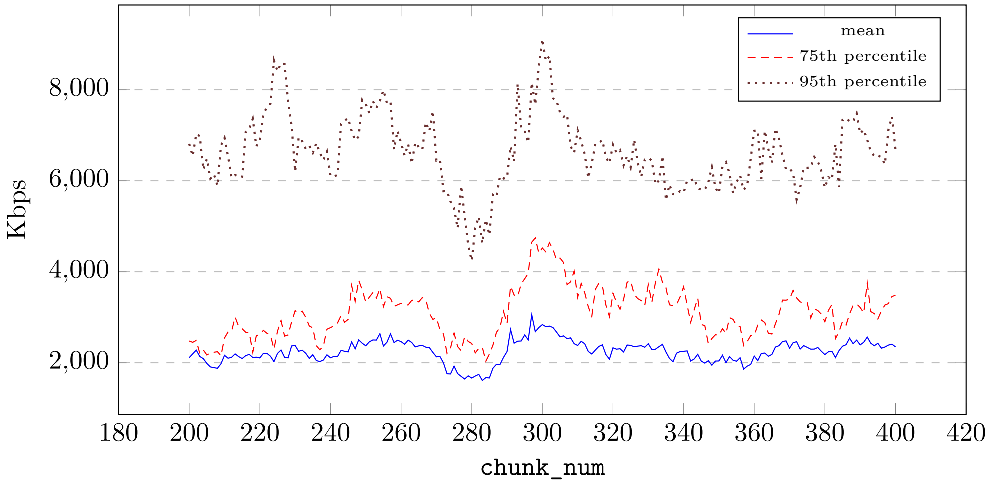
estimator is used rather than the momentary value to retain a historical value of the bandwidth, hence reduce the impact of short bursts of traffic. It is calculated using the formula $b_i = 0.8 \times b_{i-1} + 0.2 \times k/\tau$ where $k$ is the number of bits sent during the previous $\tau = 3$ seconds. This estimate is included in every receive event.

The upload bandwidth estimates for each received chunk are given in Figure 6.5. The mean upload bandwidth is above 2000 Kbps in both experiments, which represents

80% of the bit rate of unique chunks created by the broadcaster. No more than 25% of the nodes have a forwarding load of more than approximately 3000 Kbps in Figure 6.5a, and the same trend is visible in Figure 6.5b, but with a much greater variance as an effect of churn. As indicated by the 95th percentile, 5% of the nodes contribute to a big portion of the forwarding load, with estimated upload rates of more than 6000 Kbps, i.e. multiple times the bit rate of the broadcast stream. However, this remains lower than the average available upload bandwidth for broadband connections given in Table 6.1.



(a) Stable



(b) Churn

Figure 6.5: Estimated upload bandwidth.

Like for the upload bandwidth, nodes calculate an estimator of their download bandwidth by tallying bits received from peers. In Figure 6.6, the estimators for download bandwidth of nodes in the stable experiment are given. The graph shows that the estimated bandwidth is varying, and a portion of the nodes are downloading at rates up to 2800 Kbps. However, the majority of the nodes have download rates close to the rate of the media stream. An estimated download rate higher than 2500 Kbps is expected since the signaling messages in the overlay also amounts to some bandwidth.

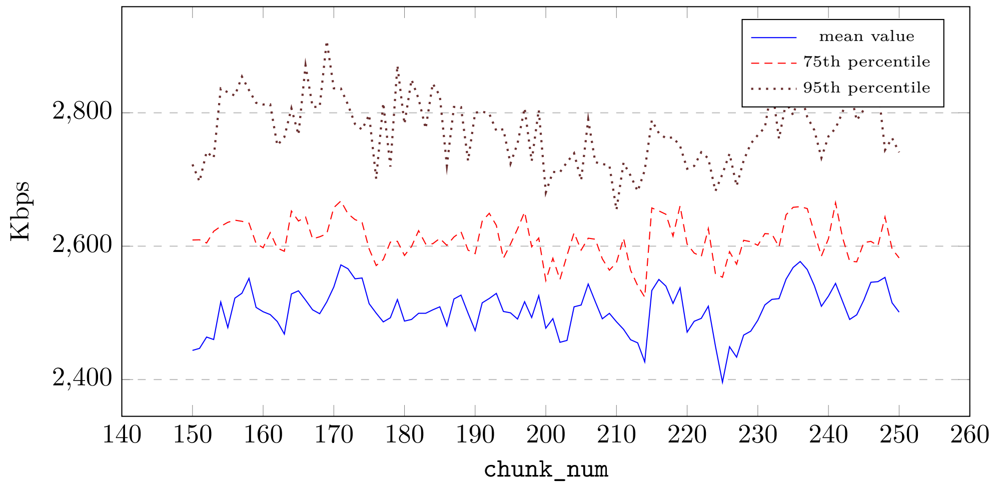


Figure 6.6: Estimated download bandwidth in a stable overlay network.

## 6.4 Summary

This chapter has described the experimental setup and given some results obtained in the two experiments conducted. Due to the limited time frame of this thesis and the amount of time required to run an experiment, a limited number of configuration have been experimented with.

The obtained results show that the bandwidth used by the nodes in the overlay network remains within reasonable limits, and are well within the values given in Table 6.1. The chunk diffusion times presented in Figure 6.3 must be seen in relation to the hop counts given in Figure 6.4 due to the low RTTs between host computers used in the experiments. An important observation is the limitations of the implemented pull policy derived from Figure 6.3. Further experiments should be conducted after these issues have been addressed in the implementation.

# Chapter 7 Discussion

This chapter discusses the results given in chapter 6 and evaluates the implementation by identifying some flaws, and it suggests ideas for future improvements.

## 7.1 The Dugnad Overlay Network

The following sections look at the overlay network in the Dugnad architecture, and outline missing features and discuss existing features that have potential for improvement.

### 7.1.1 Congestion Control

As mentioned in [MR06], congestion control on peer connections is important to efficiently share network resources with co-existing traffic. This is not included in the presented implementation, and the same goes for end-to-end flow control, as the underlying protocol used in the presented implementation is UDP. Relying on Transmission Control Protocol (TCP) to provide these mechanisms is possible for `DataChannels`, but could introduce additional delays due to protocol overhead compared to the connectionless UDP.

### 7.1.2 Semantic Coordinates

The coordinates used in the presented Dugnad implementation are static and not related to properties of the nodes they are assigned to. Adding semantics to the coordinates can be done through the addition of dimensions that represent specific properties of the node, as briefly discussed in section 4.7. These properties can be static and chosen by the RP upon assigning the node's coordinates, such as the node's position represented by latitude and longitude; properties related to the IP-address prefix of the node's public IP, and similar. The necessary information to derive these values can be included in the node's bootstrap request.

In addition to static coordinates that define relationships between nodes, dynamic dimensions can be added to the coordinates. This is partly inspired by the decentralized network coordinate system Vivaldi, where nodes measure the RTT to their peers in order to estimate the network distance to other nodes [DCKM04]. The purpose is to find a coordinates such that the Euclidean distance between the node and its peers is proportional to the network distance in terms of RTT between the parties. The basic idea from Vivaldi can be applied in Dugnad, where a peer can measure the RTT to its peers and incorporate the measurement in one dimension of the peer's virtual coordinate. Other properties that can be used as dynamic dimensions of a coordinate are the observed hop count of pieces received from peers, and properties of exchanged availability information. Also as mentioned in section 4.7, dynamic dimensions can be used to choke peers, i.e. prevent them from sending data to the node for a period of time.

The distance definition given in section 4.4 can be relaxed to incorporate weighed dimensions, hence some dimensions relating to physical properties can weigh heavier in the distance calculation than static properties that are mainly used to decide if a peer is a parent or a child. Thus, the definition of parent and child relations given in section 4.4 could be adjusted to only include the static dimensions of the coordinate decided by the RP in the bootstrap phase.

### 7.1.3 Incorporating Data-Driven Peer Selection

The results given in Figure 6.3 shows that the implementation manages to disseminate chunks to the majority of the nodes in the overlay network within a reasonable time. However, the last 5% of the nodes amount to a substantial amount of time taken to broadcast a chunk to all nodes in the overlay. Additionally, Figure 6.2 show that even when there is no churn, the current implementation fail to reach 100% of the nodes in the overlay network. This is due to an inefficient pull policy, where pull requests only contain limited availability information for a node, and is strictly used to request specific pieces.

In DONet, availability information is periodically exchanged between peers, allowing nodes to schedule pulling pieces from nodes where they know the piece is available. This approach can also be applied in Dugnad, which should effectively remove these problems that are prevalent in the current implementation.

Note that data-driven peer selection would break with the peer selection policy described in subsection 4.6.2, as the node would rely on availability information from its peers when selecting which peer to pull data from. However, in cases where multiple peers possess the same data and where dynamic coordinates are available, close peers can and should be prioritized.

### 7.1.4 Implications of Coordinate Based Peer Selection

In the implementation presented in this thesis, the broadcaster peer selection is deterministic, as the copies of each piece are always sent to the same nodes in the first hop. This is not ideal, as the current implementation rely on UDP as the underlying protocol, hence node failure can cause the broadcaster to spend a whole connection timeout delay of 8 seconds sending data to an unresponsive node. Node departures can be detected faster using TCP as the underlying protocol, as the lack of response from the remote party will be detected before the connection times out if transferred data is not acknowledged. However, lowering the connection timeout delay would also achieve this.

Another problem with the broadcaster selecting a few nodes for which to transfer data is that these nodes may not be able to provide the forwarding capacity required for the broadcast, thus creating a bandwidth bottleneck. As mentioned in section 3.5 based on the discussion in [MP10], tit-for-tat schemes, i.e. incentive based schemes, do not work for the first hop, as the broadcaster will not have anything to download. Hence who the broadcaster forwards the initial data to can make a big difference, but making the right choice is hard. Data-driven peer selection is also not applicable here since all nodes are equally deprived of the new chunks that are originating at the broadcaster.

A possible improvement would be to spread the pieces of a chunk to multiple nodes, such that no single node is in possession of a complete chunk after the first hop. This way the nodes in the first hop are not required to have an upload capacity equal to or greater than the rate of the media stream to not be bottlenecks, as they are only required to forward parts of the stream. However, due to the push component of Dugnad, the broadcaster should forward data to the nodes that are closest to the origin in the virtual coordinate space.

Ideally, benign nodes with high forwarding capacities should be awarded coordinates close to the broadcaster, but these properties are not possible to see in a node a priori. It has been seen that tit-for-tat schemes do not apply for the first hop, but a distributed way of finding high-capacity nodes could allow the RP to move the node closer to the broadcaster, hence allow for an authenticated change of coordinates. Details on how to make this possible have not been studied in this thesis.

### 7.1.5 Scoring Peers

The presented implementation have a very simple scheme of scoring peers by punishing nodes forwarding redundant data, as described in subsection 5.5.3. A potentially more efficient solution would be to detect logical shortcuts by comparing hop counts of received chunks on connections as well as exchanging and comparing availability data to a greater extent. This way, forwarding redundant data can be avoided without taking the extreme measure of the other party closing the connection.

The choke mechanism found in BitTorrent [Coh08] can be realized using dynamic coordinates as discussed in subsection 7.1.2, thus prevent nodes from pushing redundant pieces. A peer score can still be incorporated in a tit-for-tat scheme between peers, thus allowing nodes to discover more efficient connections by freeing up space for new peers.

## 7.2 Scalability of Dugnad

The one factor limiting the scalability of Dugnad is the load imposed by WebSocket management in the RP. Traditional P2P systems require the RP to keep a list of IP addresses and port numbers of nodes; WebRTC requires a proxy with an established connection to both parties prior to connection establishment between the nodes. This is the biggest differentiating factor between traditional P2P systems and the WebRTC environment.

Dugnad requires the RP to keep open connections to all nodes in the overlay network. However, this constraint may be relaxed, as the RP can keep a selection of the nodes at all times. Nodes that require the proxy services of the RP will initiate a connection, hence reopen the WebSocket if it was closed; the RP can then forward communication between the node and one of the other open connections. Ultimately, keeping all connections open should be feasible and more scalable than serving traditional unicast streams in terms of bandwidth. Only signaling is transmitted by the RP, and the RP can be spread over multiple servers and load balancing strategies can be applied, allowing the system can handle big viewer numbers at a relatively low cost. Specifics in this regard are outside the scope of this thesis.

As discussed in section 4.8, it is technically possible for a node in the overlay network to assist in connection establishment between its peers without involving the RP, and an example of such system is studied in [VWS13]. We have also seen how this may introduce a vulnerability to the system for which malicious nodes can take advantage to obtain a better service and possibly deny service to others. However, it remains an open question if the severity of this vulnerability justifies the choice to not take advantage of these capabilities, as it would greatly improve the scalability of the system in terms of signaling load on the RP.

## 7.3 Security

The following sections discuss some security considerations and implications of Dugnad, and web-based P2P applications in general.

### 7.3.1 Access Control

A P2P system based on WebRTC requires an established RP for initial connection establishment. This allows the RP to act as an access control point, enforcing an access control polity to given broadcast channels. However, simply denying initial signaling is not sufficient to keep unwanted nodes out of the overlay network, as the clients have full control of the application they run, hence nodes can theoretically establish connections through third party signaling channels provided by browser extensions which do not enforce the access control policy. For the node to extend its footprint in the overlay network by establishing additional connections harder in Dugnad; benign nodes will only establish connections where the signaling is received from the RP, disallowing nodes in the overlay network to act as a proxy for connection establishment between peers. Hence a node joining the network through a third party signaling channel and a cooperating malicious node will not be able to connect to benign nodes in Dugnad as long as the connection establishment restrictions are enforced.

### 7.3.2 Data Integrity

Peers are in full control of the code running in their browser, and could chose to forward their own content instead of the video stream provided by the broadcast source. All it takes is for someone to make a browser extension and replace some of the client side Java Script, making such attacks easy to perform with minimal knowledge. Using public key cryptography where all clients download the broadcaster's public-key certificate from the trusted RP upon joining the overlay network can combat this. The broadcaster can then sign all pieces and include the signature in the broadcast data to enable clients to verify the origin and integrity of the received pieces. However, this would impose a significant load on both the viewers and the broadcaster as they would have to apply asymmetric cryptography on every piece of data for every hop which, makes this a substantial cost.

A less invasive alternative in terms of computation load is to only sign complete chunks. However, this suggests that if a malicious node forwards a single piece where some bits are flipped, all viewers receiving that copy will be unable to play the chunk once assembled, as the signature will not match. The viewers cannot know which piece is erroneous, thus making this a very low effort denial of service. For this reason this scheme is deemed impractical, and further study of data integrity remains a problem for future work.

## 7.4 Deployment Cost

Unlike traditional P2P applications, Dugnad is designed to fit in a web page and run within a web browser. This means that the cost of making changes to the implementation is very low, as all users of the application will download the latest version every time they load the web page.

# Chapter 8 Concluding Remarks

## 8.1 Conclusion

In this thesis we have seen that it is indeed possible to build a P2P live video streaming application on top of WebRTC. The Dugnad architecture has been proposed for this purpose, and some refinements have been suggested after experimenting with an implementation of the architecture. We have seen that the biggest difference between a traditional P2P application and one based on WebRTC is the establishment of connections between peers, which in the case of Dugnad impose some undesirable load on the RP.

The experiments conducted have shown that the currently implemented diffusion scheme is not satisfactory as it does not reliably manage to broadcast chunks of data to all nodes in the overlay network. However, from Figure 6.3 it is clear that 95% of the viewer receive their required pieces in a short amount of time. We have seen how these issues can be addressed through incorporating a data-driven peer selection in a revised pull policy.

Users of traditional television tend to switch between channels; this is a challenge for a P2P-based system, taking into account the relatively long start-up delays in mesh-pull protocols [HLL$^+$07]. In this thesis we have seen how use of virtual coordinates assigned by a trusted RP dictate the relation between peers to create an implicit structure in the overlay network. This allows nodes to push data in an overlay network without cycles, thus the network should be able to quickly disseminate rare pieces.

In a world where people upload pictures and video on a daily basis, Internet Service Providers (ISPs) and mobile operators provide increased upload bandwidths in their subscriptions. The premise for P2P technologies to work is present, even for the increasing number of mobile clients. Thus P2P live video streaming applications built on top of WebRTC should have every chance at being successful in the future, as potential users are capable of sharing an increasing amount of bandwidth.

## 8.2 Future Work

Future work should revise the RP in order to introduce static semantic dimensions to the coordinates. Additionally, dynamic coordinate dimensions should be added to enable more efficient ways to choke peers performing redundant pushes and to motivate cooperation between nodes with short network distances.

Future implementations should incorporate a DONet like pull policy, hence availability information should be exchanged between peers and used actively in the peer-selection for a pull request. Looking at ways in which high capacity nodes can be promoted to more efficiently serve the overlay network should also be subject to future work.

An issue that is not studied in this thesis is the recovery of an overlay network after a RP failure. This could be addressed in future work to improve on the dependability of the architecture.

Due to time limitations, this thesis was unable to study the effects of changing the desired peer count in the Dugnad architecture. Additionally, experimenting with bandwidth-limited nodes was not done due to the nature of the experiment setup. Overall, a very limited number of configurations have been extensively tested in the current implementation, as the presented experiments revealed some significant issues. With a revised implementation, or a simulation effort, this can be amended in future work.

# References

[ALMC14] Vasco Amaral, Solange Rito Lima, Telma Mota, and Paulo Chainho. Exploring webrtc technology for enhanced real-time services. In *New Perspectives in Information Systems and Technologies, Volume 2 [WorldCIST'14, Madeira Island, Portugal, April 15-18, 2014]*, pages 43–52, 2014.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[Alv14] Harald Alvestrand. Overview: Real time protocols for browser-based applications. Internet-Draft draft-ietf-rtcweb-overview-13, IETF Secretariat, November 2014. http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-overview-13.txt.

[BBJ15] Adam Bergkvist, Daniel C. Burnett, and Cullen Jennings. Webrtc 1.0: Real-time communication between browsers. Technical Report http://www.w3.org/TR/2015/WD-webrtc-20150210/, W3C, April 2015.

[BMM+08] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. Epidemic live streaming: optimal performance trade-offs. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2008, Annapolis, MD, USA, June 2-6, 2008*, pages 325–336, 2008.

[CBW15] Aaron Colwell, Adrian Bateman, and Mark Watson. The websocket api. Technical Report http://www.w3.org/TR/2015/CR-media-source-20150331/, W3C, March 2015.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony I. T. Rowstron, and Atul Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, pages 298–313, 2003.

[Coh08] Bram Cohen. The bittorrent protocol specification. http://www.bittorrent.org/beps/bep_0003.html, 2008. Accessed: 2015-06-07.

[DCKM04] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 30 - September 3, 2004, Portland, Oregon, USA*, pages 15–26, 2004.

[Dee89] Steve Deering. Host extensions for ip multicasting. STD 5, IETF, August 1989. https://tools.ietf.org/html/rfc1112.

[Hic12] Ian Hickson. Media source extensions. Technical Report http://www.w3.org/TR/2012/CR-websockets-20120920/, W3C, September 2012.

[HLL$^+$07] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and Keith W. Ross. A measurement study of a large-scale P2P IPTV system. *IEEE Transactions on Multimedia*, 9(8):1672–1687, 2007.

[Ive10] Villy B Iversen. Teletraffic engineering and network planning. *DTU Course*, 34340:554, 2010.

[MMR10] R. Mahy, P. Matthews, and J. Rosenberg. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). RFC 5766, IETF, April 2010. https://tools.ietf.org/html/rfc5766.

[MP10] Fabien Mathieu and Diego Perino. On resource aware algorithms in epidemic live streaming. In *22nd International Teletraffic Congress, ITC 2010, Amsterdam, The Netherlands, September 7-9, 2010*, pages 1–8, 2010.

[MR06] Nazanin Magharei and Reza Rejaie. Understanding mesh-based peer-to-peer streaming. In *Network and Operating System Support for Digital Audio and Video, 16th International Workshop, NOSSDAV 2006, Newport, Rhode Island, USA, November 22-23, 2006, Proceedings*, page 10, 2006.

[PPKB07] Fabio Pianese, Diego Perino, Joaquín Keller, and Ernst W. Biersack. PULSE: an adaptive, incentive-based, unstructured P2P live streaming system. *IEEE Transactions on Multimedia*, 9(8):1645–1660, 2007.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, pages 329–350, 2001.

[RMMW08] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for nat (stun). RFC 5389, IETF, October 2008. https://tools.ietf.org/html/rfc5389.

[RTCa] Rtcdatachannel. https://developer.mozilla.org/en-US/docs/Web/API/RTCDataChannel. Accessed: 2015-05-05.

[RTCb] Rtcpeerconnection. https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection. Accessed: 2015-05-05.

[SGGS09] Salvatore Spoto, Rossano Gaeta, Marco Grangetto, and Matteo Sereno. Analysis of pplive through active and passive measurements. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pages 1–7, 2009.

[SHM07] Sujay Sanghavi, Bruce Hajek, and Laurent Massoulié. Gossiping with multiple messages. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA*, pages 2135–2143, 2007.

[VWS13] Christian Vogt, Max Jonas Werner, and Thomas C. Schmidt. Leveraging webrtc for P2P content distribution in web browsers. In *2013 21st IEEE International Conference on Network Protocols, ICNP 2013, Göttingen, Germany, October 7-10, 2013*, pages 1–2, 2013.

[ZLLY05] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005, Miami, FL, USA*, pages 2102–2111, 2005.

[ZLZ$^+$05] Li Zhao, Jian-Guang Luo, Meng Zhang, Wen-Jie Fu, Ji Luo, Yi-Fei Zhang, and Shi-Qiang Yang. Gridmedia: A practical peer-to-peer based live video streaming system. In *IEEE 7th Workshop on Multimedia Signal Processing, MMSP 2005, October 30 2005 - November 2 2005, Shanghai, China*, pages 1–4, 2005.