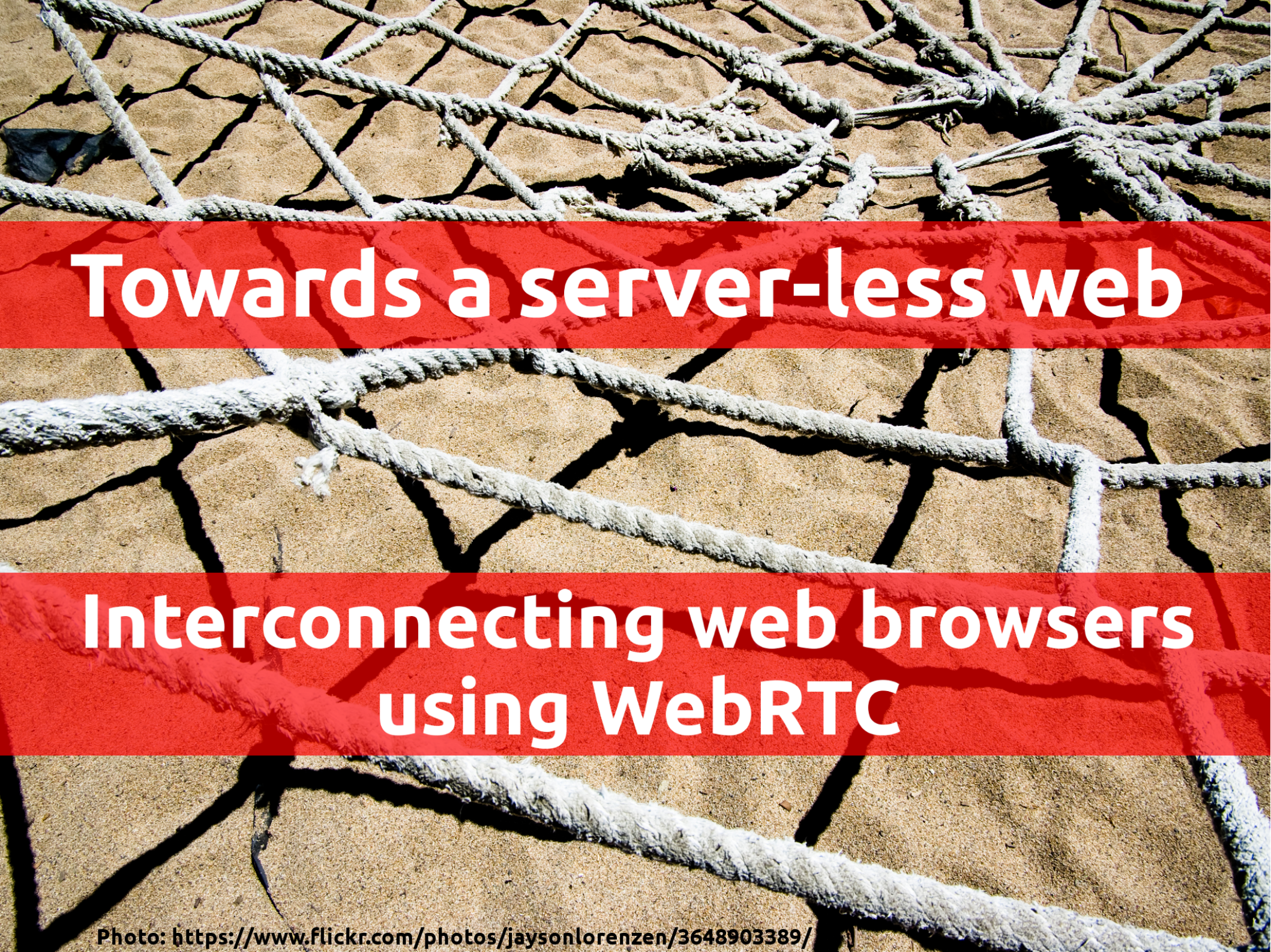
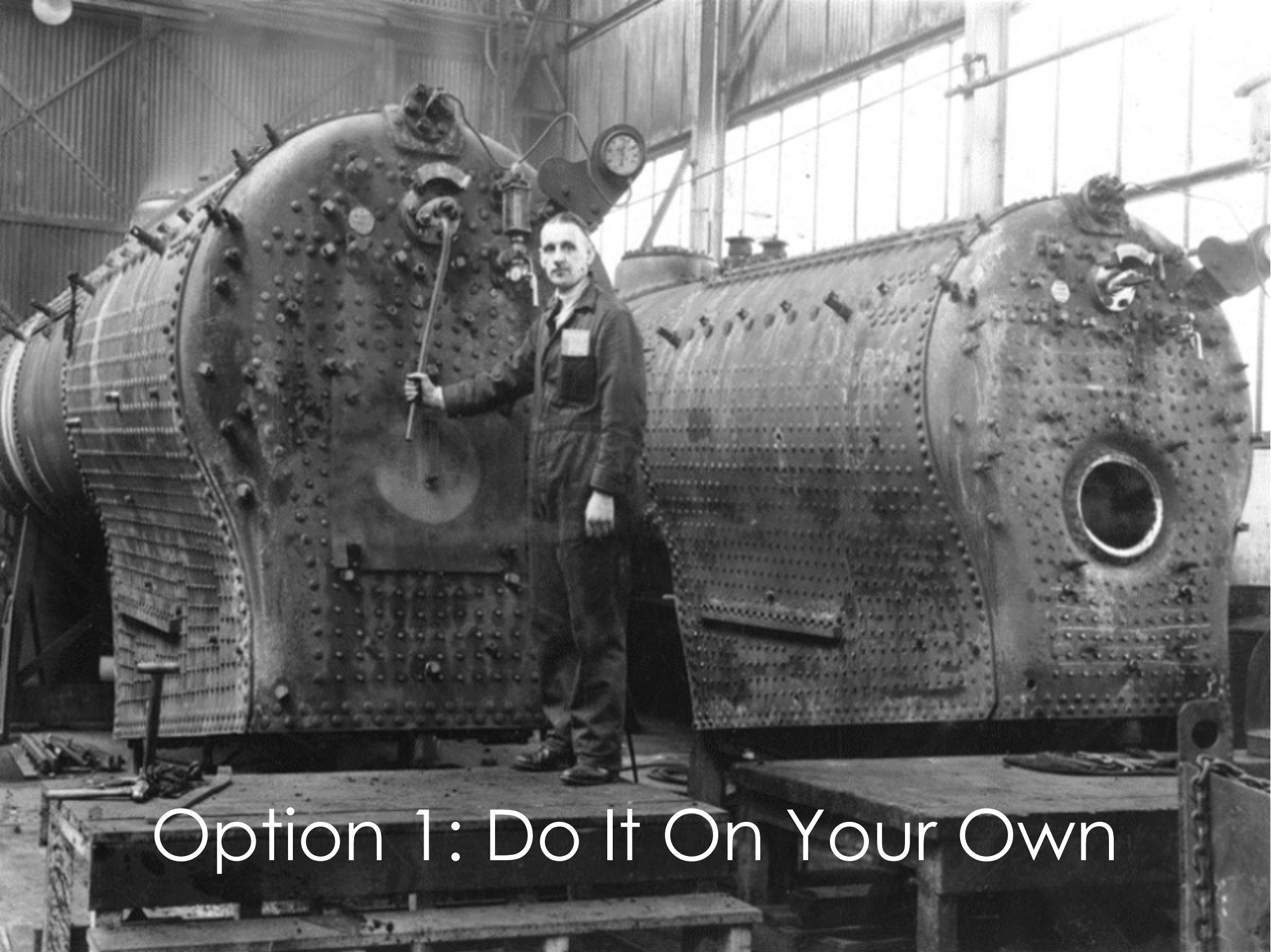

Towards a server-less web
Interconnecting web browsers using WebRTC
Photo: https://www.flickr.com/photos/jaysonlorenzen/3648903389/



Option 1: Do It On Your Own

Option 2: Let Someone Else Do It

Migration: Addressing Problem

Oops!
You found a
Dead Link

Take back Control

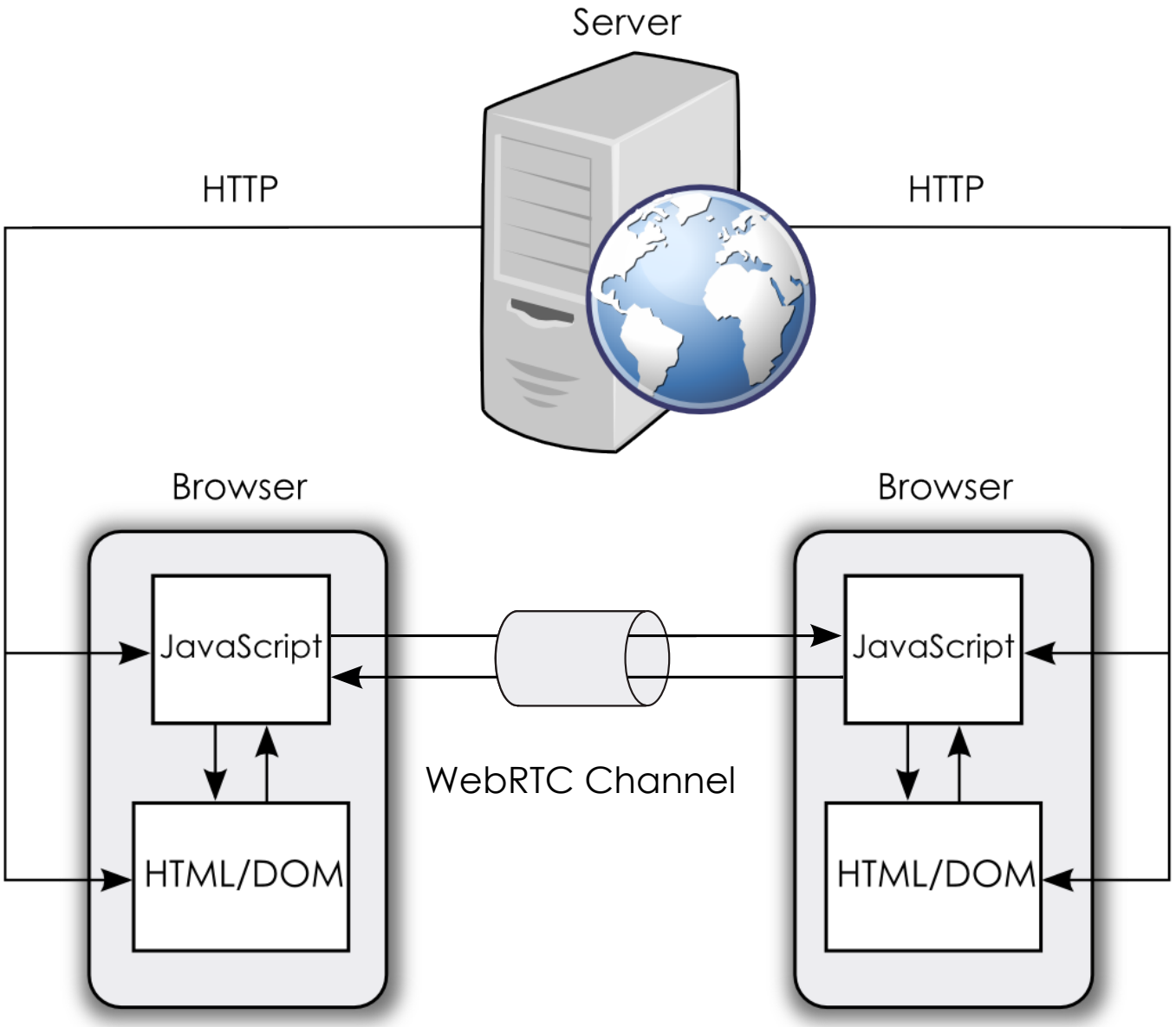
What is WebRTC?

WebRTC is a free, open **project** that provides browsers and mobile applications with **Real-Time Communications** (RTC) capabilities via simple APIs.

> WebRTC (Web Real-Time Communication) is an **API** definition drafted by the World Wide Web Consortium (W3C) that supports **browser-to-browser applications** for **voice calling, video chat, and P2P file sharing** without the need of either internal or external plugins.

WebRTC is a **communications standard** developed by the W3C in close cooperation with the RTCWeb standard developed by the IETF. RTCWeb functions at a **lower protocol layer**; WebRTC enables the embedding of this functionality in applications and websites. The protocol is commonly used to support **voice or video chat** between peers.

This is WebRTC



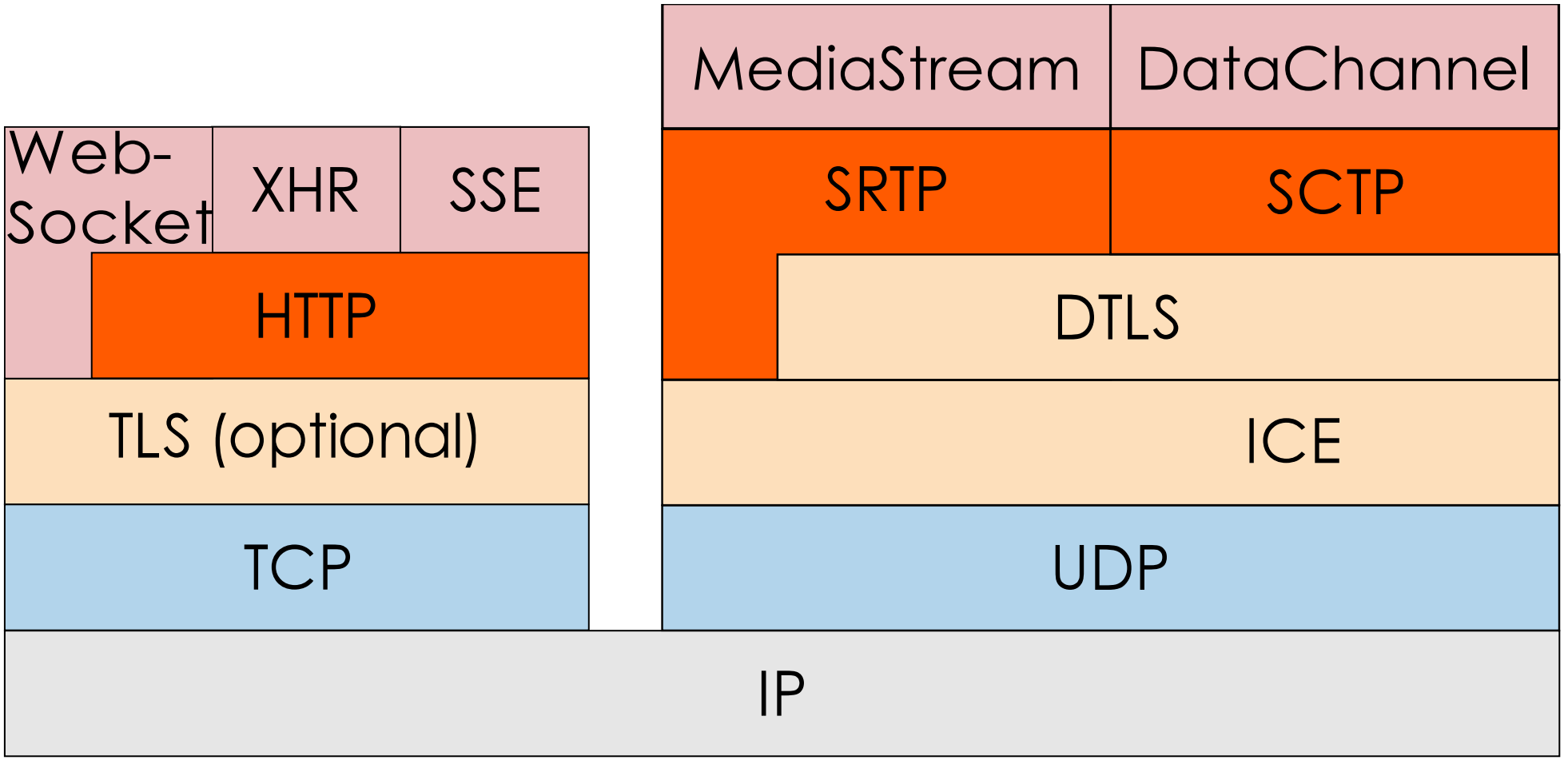
Networking in Browsers

	Asynchronous	Architecture	Communication
HTTP	no	Client/Server	Unidirectional
Server-sent Events	yes	Client/Server	unidirectional
WebSocket	yes	Client/Server	bidirectional
WebRTC	yes	Peer-to-Peer	bidirectional

This is WebRTC

- Enables direct communication between Web Browsers
- NAT Traversal
- No intermediary server necessary
- Secure RTP for audio/video data
- SCTP over DTLS for generic data
- Signaling via SDP offer/answer
- Optional source authentication

The Protocol Stack



The API: Basics

```
// Alice
var pc = new window.RTCPeerConnection();
channel.onmessage = function(msg){
  pc.setRemoteDescription(msg);
};
function createOfferCallback(offer) {
  pc.setLocalDescription(offer);
  channel.send(offer);
}
pc.createOffer(createOfferCallback);
```

```
// Bob
var pc = new window.RTCPeerConnection();
function createAnswerCallback(answer) {
  channel.send(answer);
}
channel.onmessage = function(msg){
  pc.setRemoteDescription(msg);
  pc.createAnswer(createAnswerCallback);
};
```

The API: Data Channels

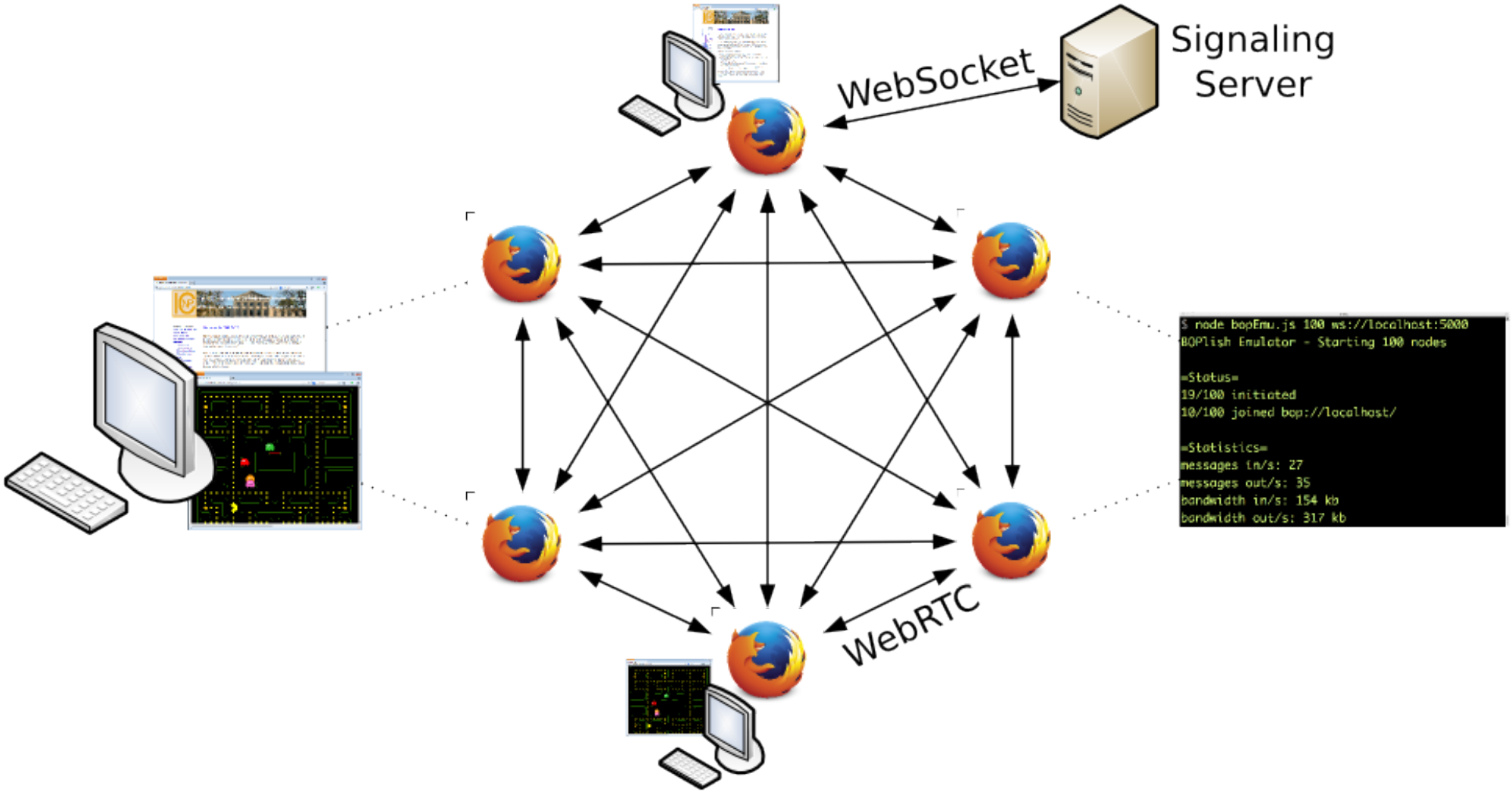
```
// Alice
var dc = pc.createDataChannel();
dc.onopen = function() {
  dc.send('hi bob');
};
```

```
// Bob
pc.ondatachannel = function(evt) {
  var dc = evt.target;
  dc.onmessage = function(msg) {
    alert(msg); // alerts 'hi bob'
  };
};
```

The API (on the wire)

```
1 v=0 // protocol version
2 o=Mozilla-SIPUA-33.0 14557 0 IN IP4 0.0.0.0 // origin
identifier
3 s=SIP Call // session name
4 t=0 0 // may indicate start/stop times; not used in WebRTC
5 a=ice-ufrag:c43d936f // ICE parameter
6 a=ice-pwd:0a0435ce7407bfc0ec43a953278916c7 // ICE parameter
7 a=fingerprint:sha-256 <omitted> // DTLS parameter
8 m=application 49687 DTLS/SCTP 5000 // Request for Data Channel
9 c=IN IP4 84.130.199.184 // Connection Endpoint (overwritten by
ICE)
10 a=sctpmap:5000 webrtc-datachannel 16 // Data Channel parameter
11 a=setup:actpass // Identifies the offerer, wait for answer
12 // ICE candidates following:
13 a=candidate:0 1 UDP 2130379007 192.168.0.20 49687 typ host
14 a=candidate:1 1 UDP 1694236671 84.130.199.184 49687 typ srflx
raddr 192.168.0.20 rport 49687
```

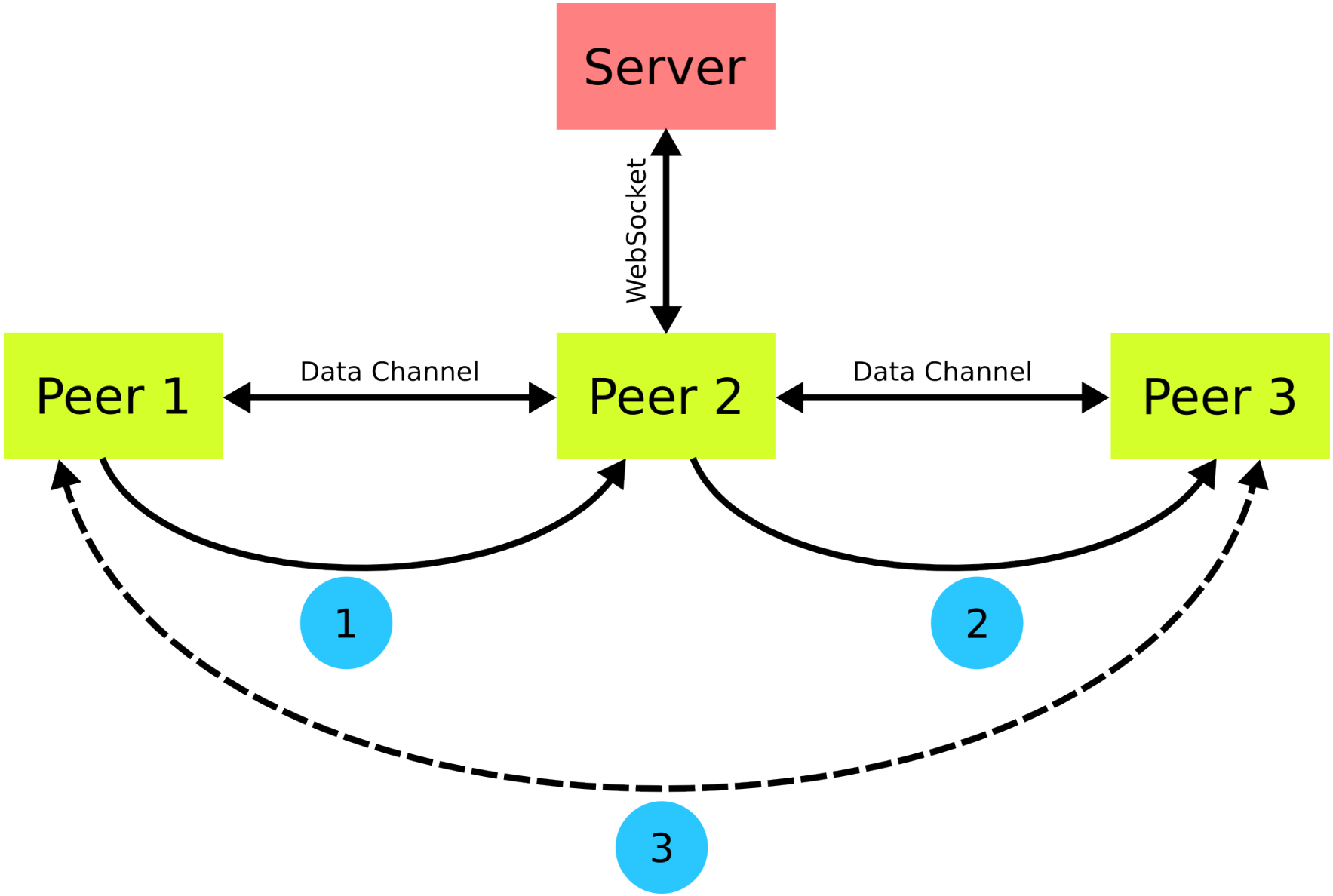
BOPlish: Overview



BOPlish: What do we need?

- Routing of Data
- Content Names and Name Resolution
- Simple API

Routing


Server
WebSocket
Peer 1
Data Channel
Peer 2
Data Channel
Peer 3
1
2
3

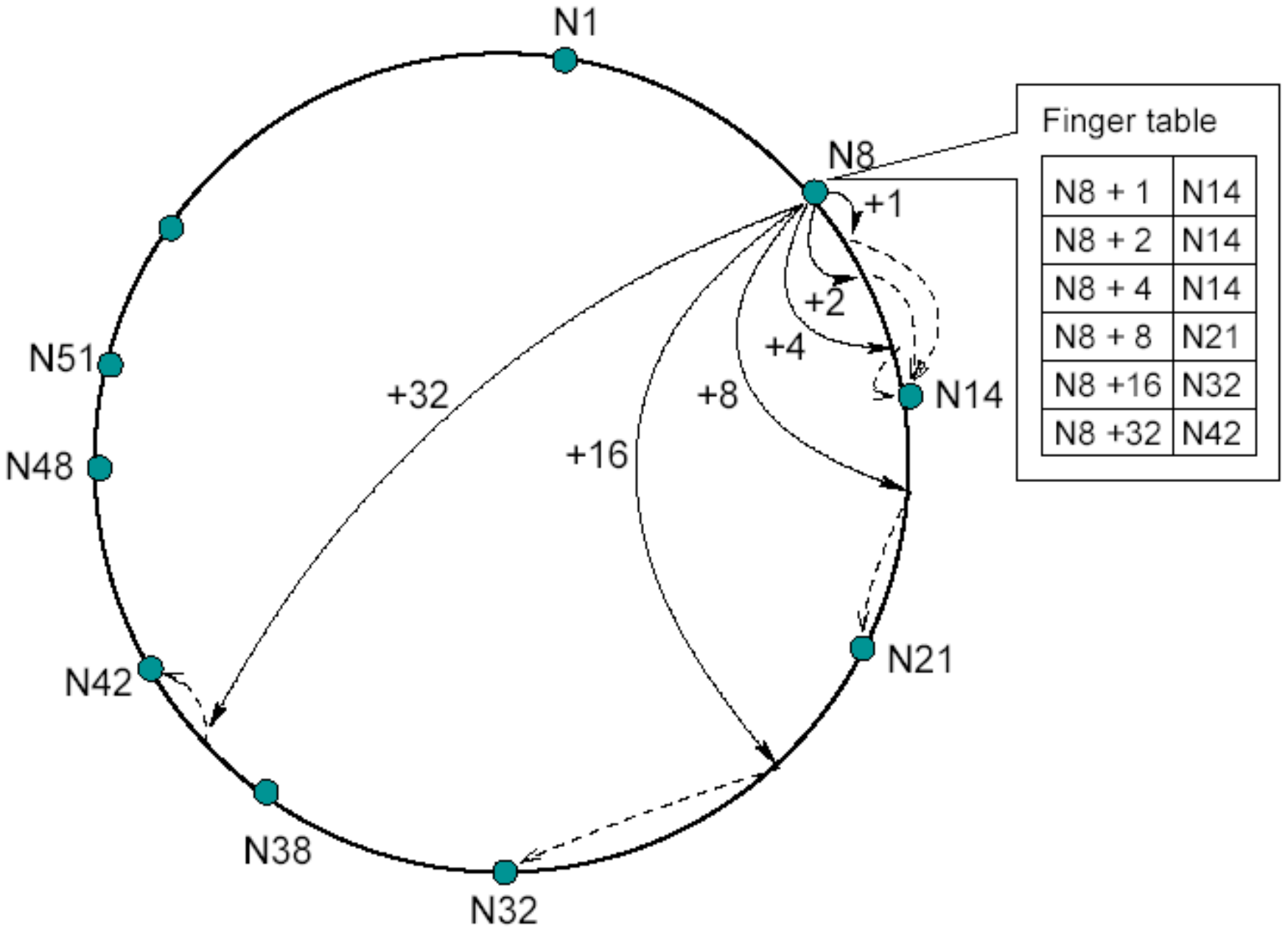
Content Names

- Location-independent names
- Easy replication and migration
- URIs, the common scheme for names on the Web:

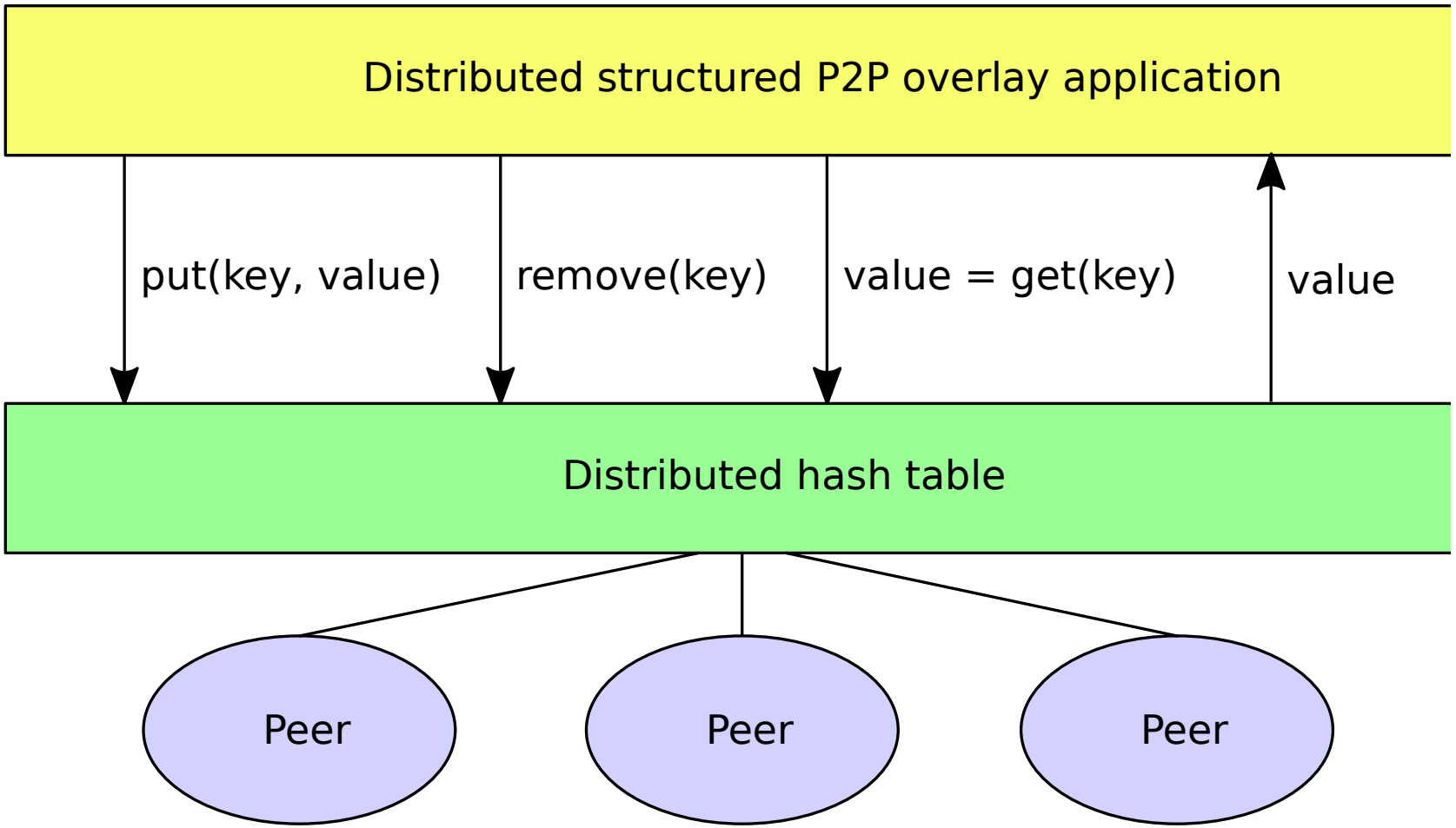
```
bop:username@idp:protocol[/path[?parameters]]
```

Name Resolution: DHT

a distributed key/value store

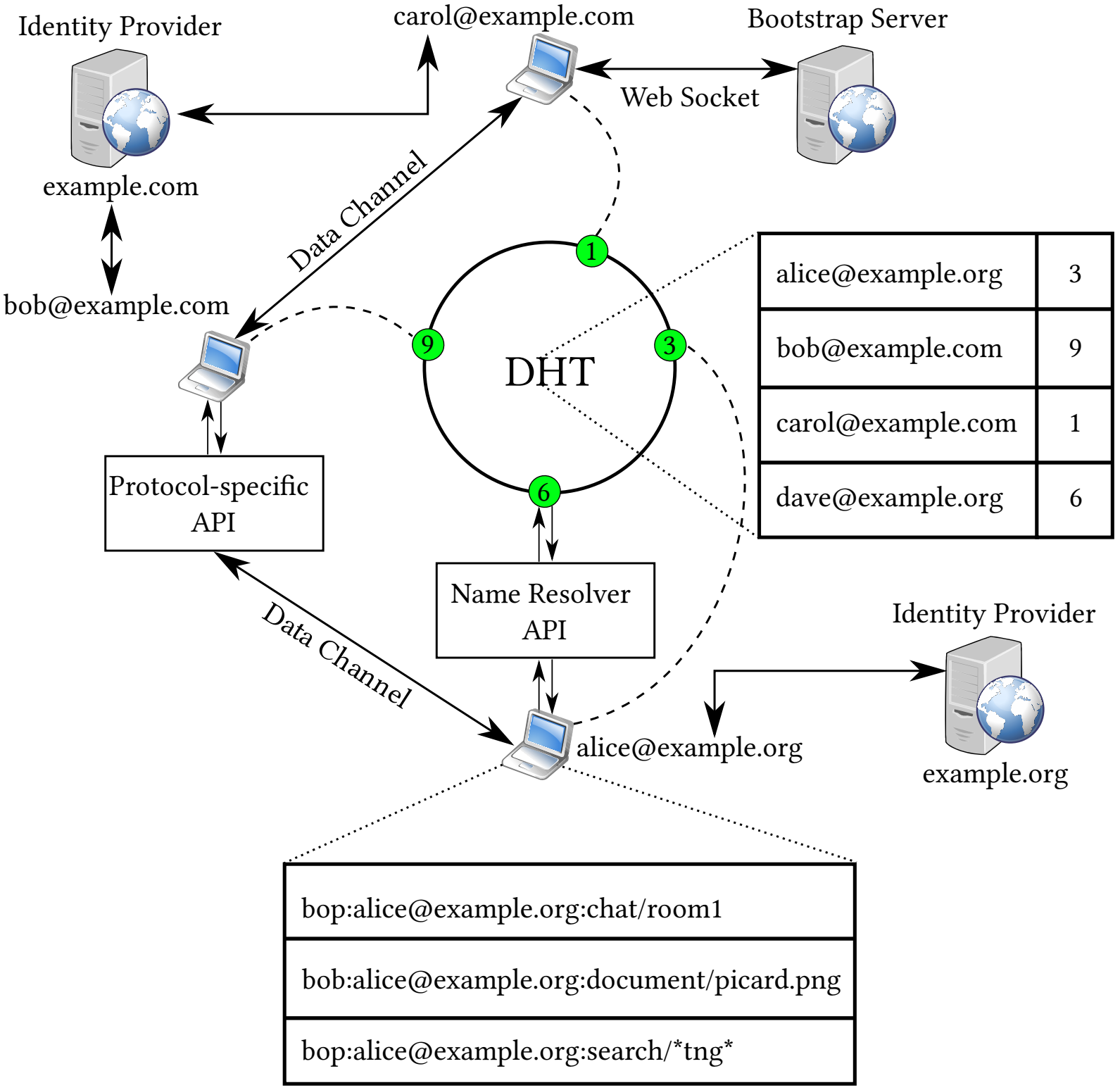


Name Resolution: API



Simple API

```
var client = new BOPlishClient("ws://boplish.com",
    function onconnect(){...},
    function onerror(){...});
var protocol = client.registerProtocol('groupchat');
protocol.onmessage = function onmessage(msg) {...};
```


Identity Provider
carol@example.com
Bootstrap Server
Web Socket
example.com
Data Channel
bob@example.com
1
9
3
DHT
6
alice@example.org
3
bob@example.com
9
carol@example.com
1
dave@example.org
6
Protocol-specific API
Name Resolver API
Data Channel
Identity Provider
alice@example.org
example.org
bop:alice@example.org:chat/room1
bob:alice@example.org:document/picard.png
bop:alice@example.org:search/*tng*

Use Cases

- Document Sharing
- Content Search
- Real-time Chat

Use Cases: Document Sharing

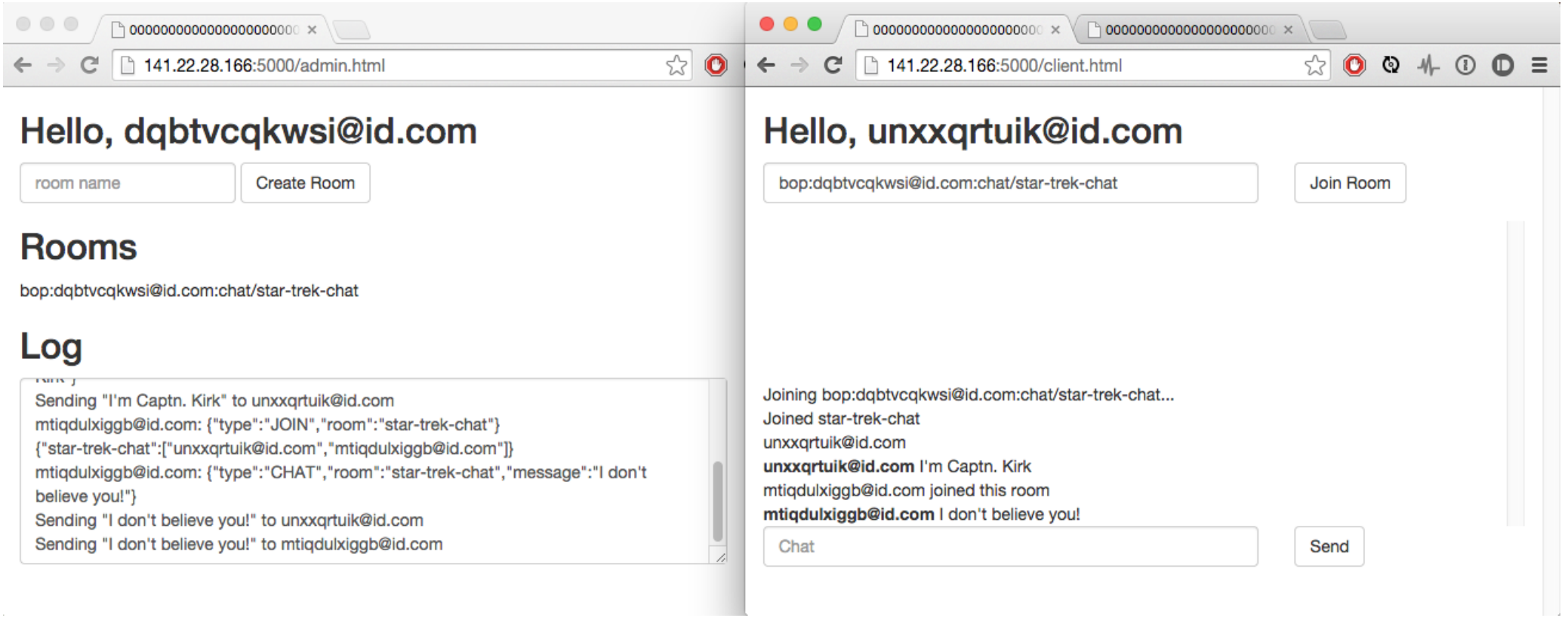
“Every time you email a file to yourself so you can pull it up on your friend's laptop, Tim Berners-Lee sheds a single tear.”

```
bop://max@example.org/family.png?\
csum=sha256:a2bd. . .
```

Use Cases: Content Search

```
bop://max@example.org/Music/*johnossi*
```

Use Cases: Real-Time Chat



`bop:alice@example.org:chat/room1`

Ressources

Project Home

https://github.com/boplish/

Core Library

https://github.com/boplish/core/

Demo Applications

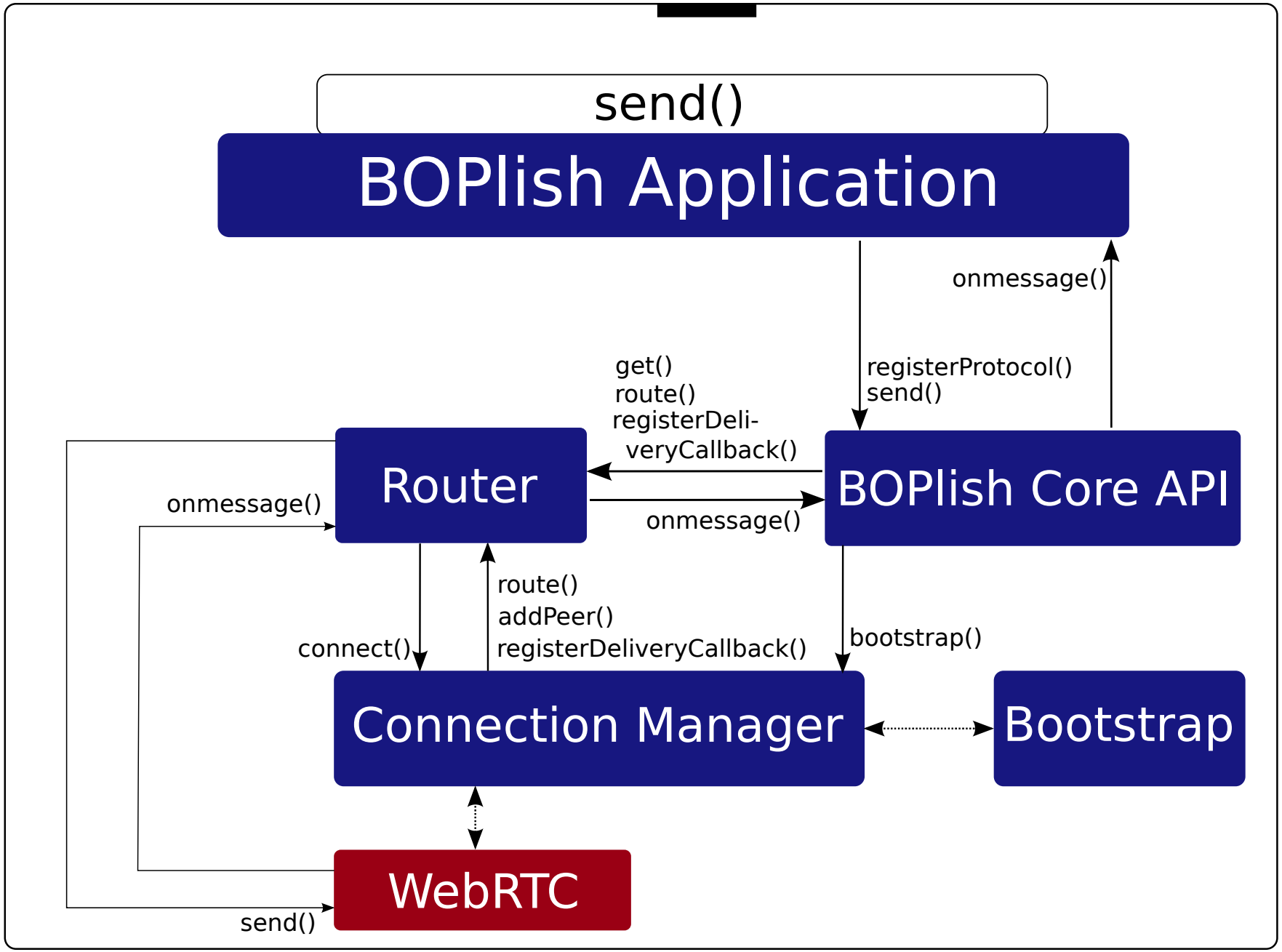
https://github.com/boplish/demos/

Recap/Outlook

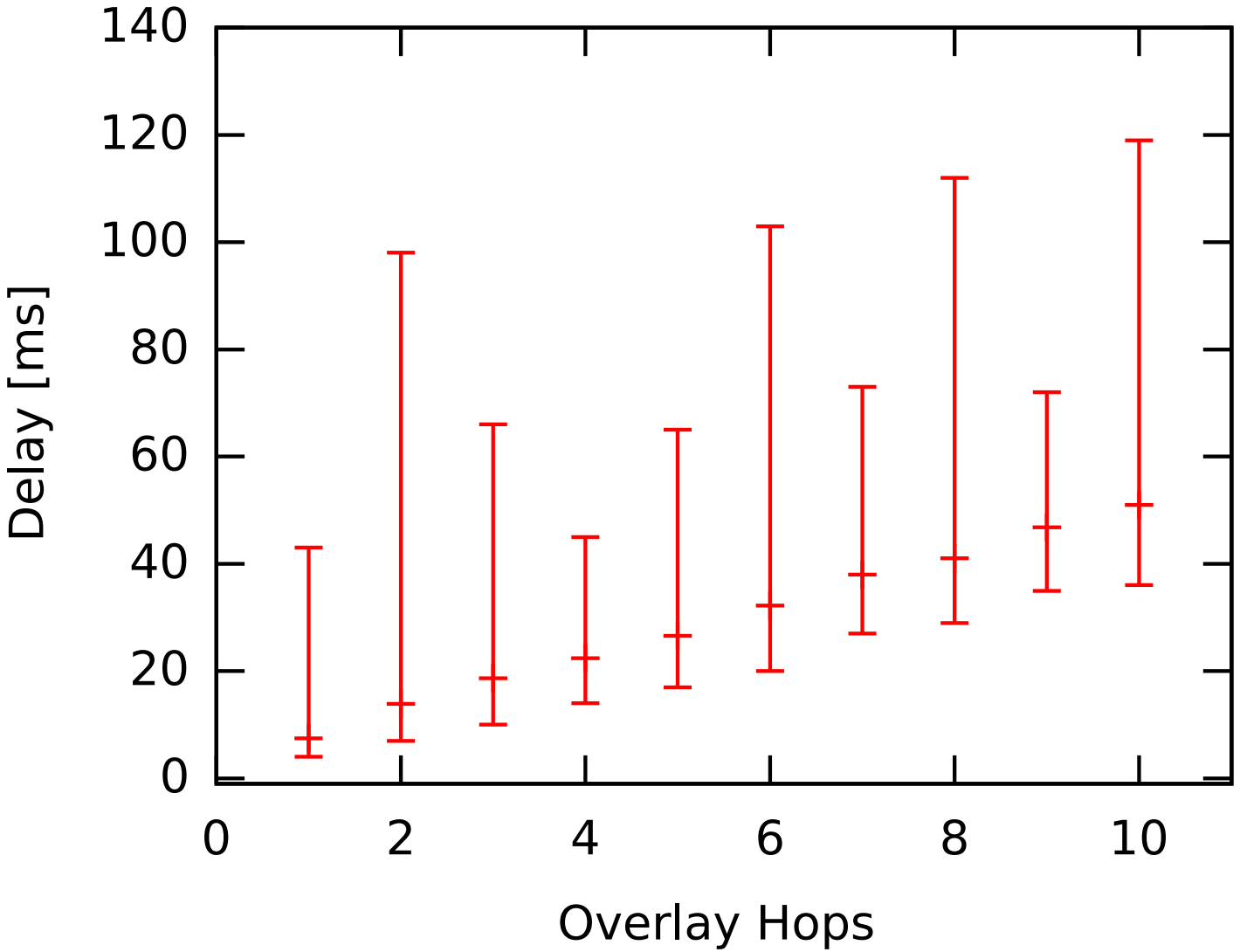
- BOPlish provides a framework for distributed content communities
- It's as simple as pulling in boplish.js
- WebRTC is cumbersome at times
- TODO: mobility, content offloading, network/P2P stability, security

BACKUP

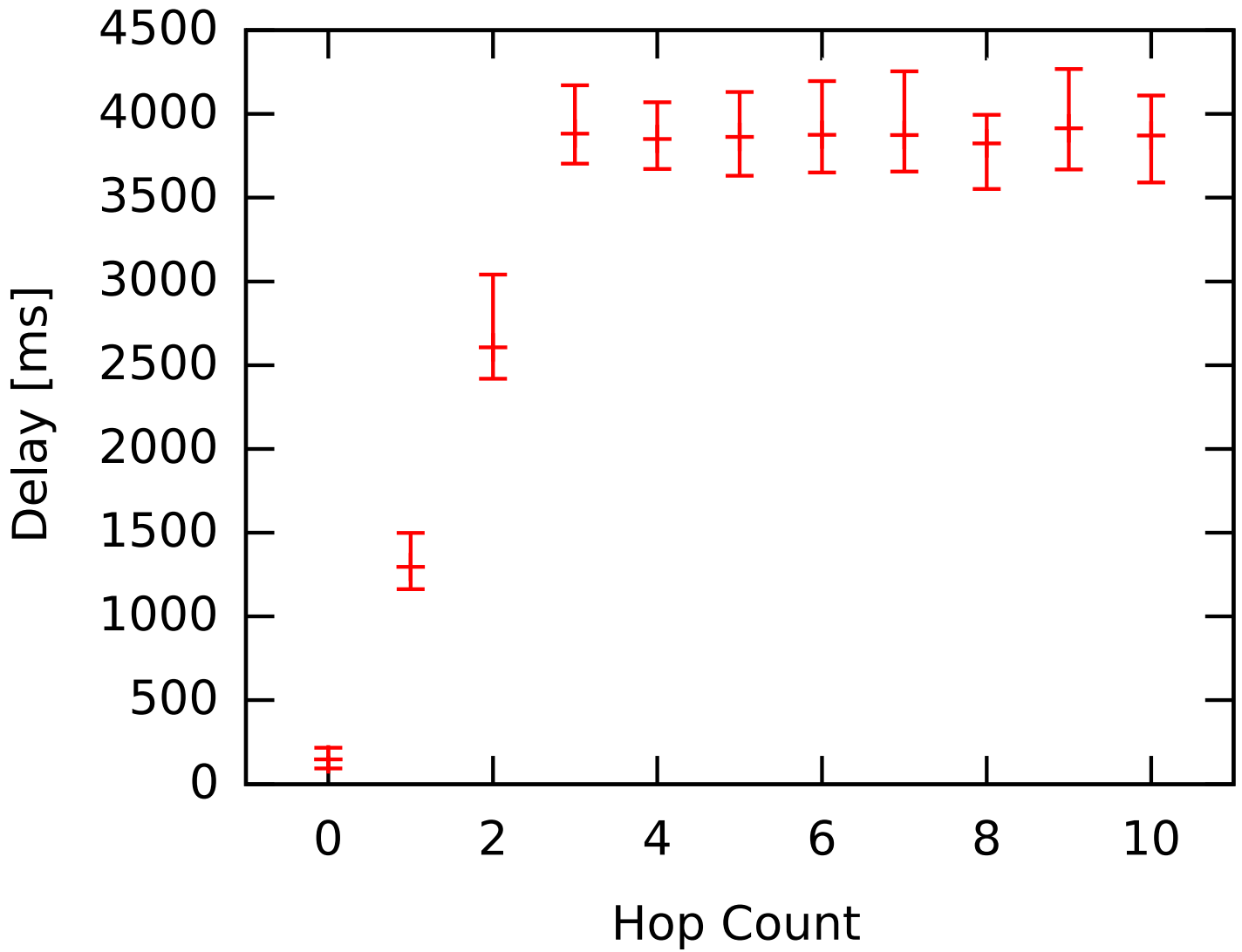
Architecture



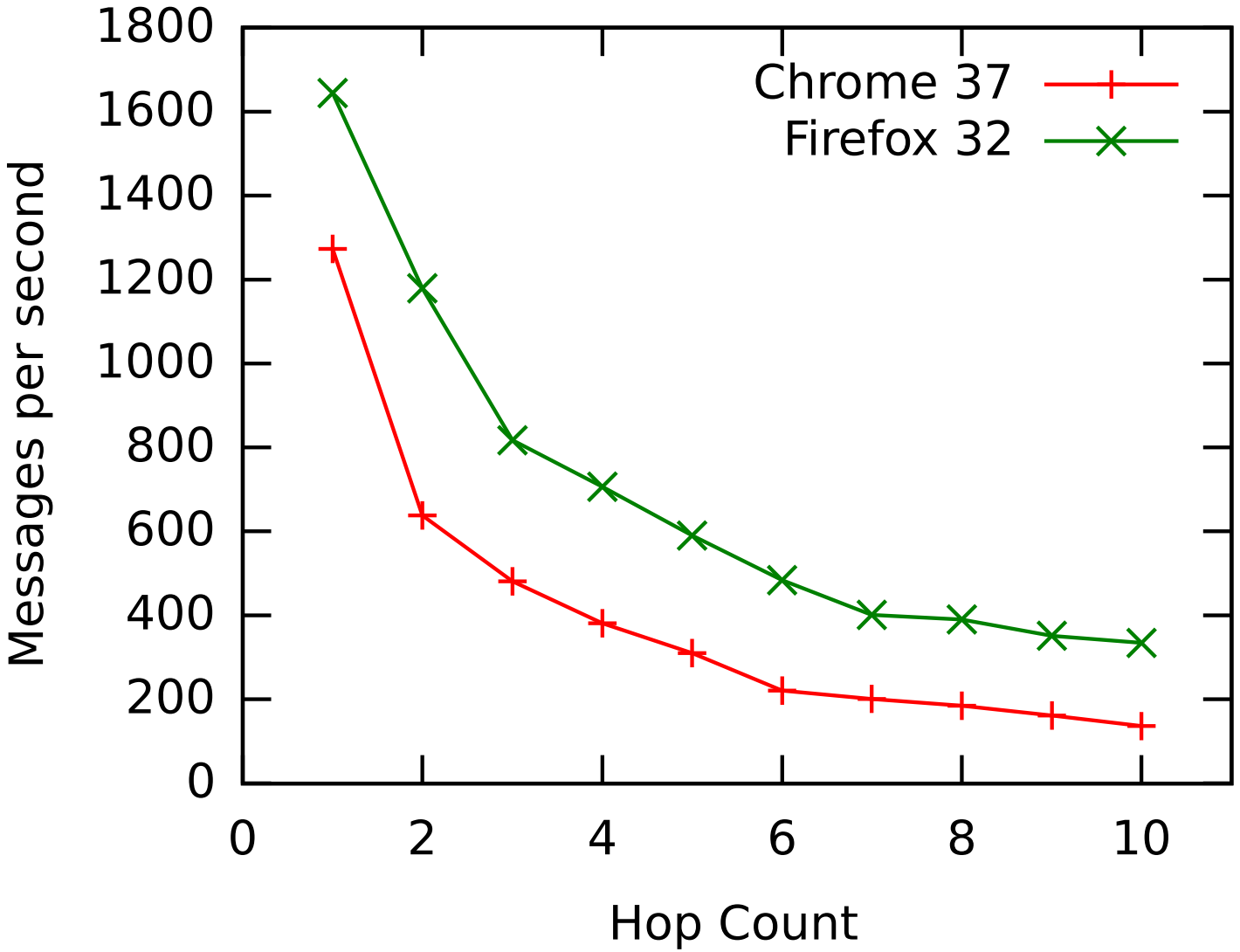
Lookup Performance



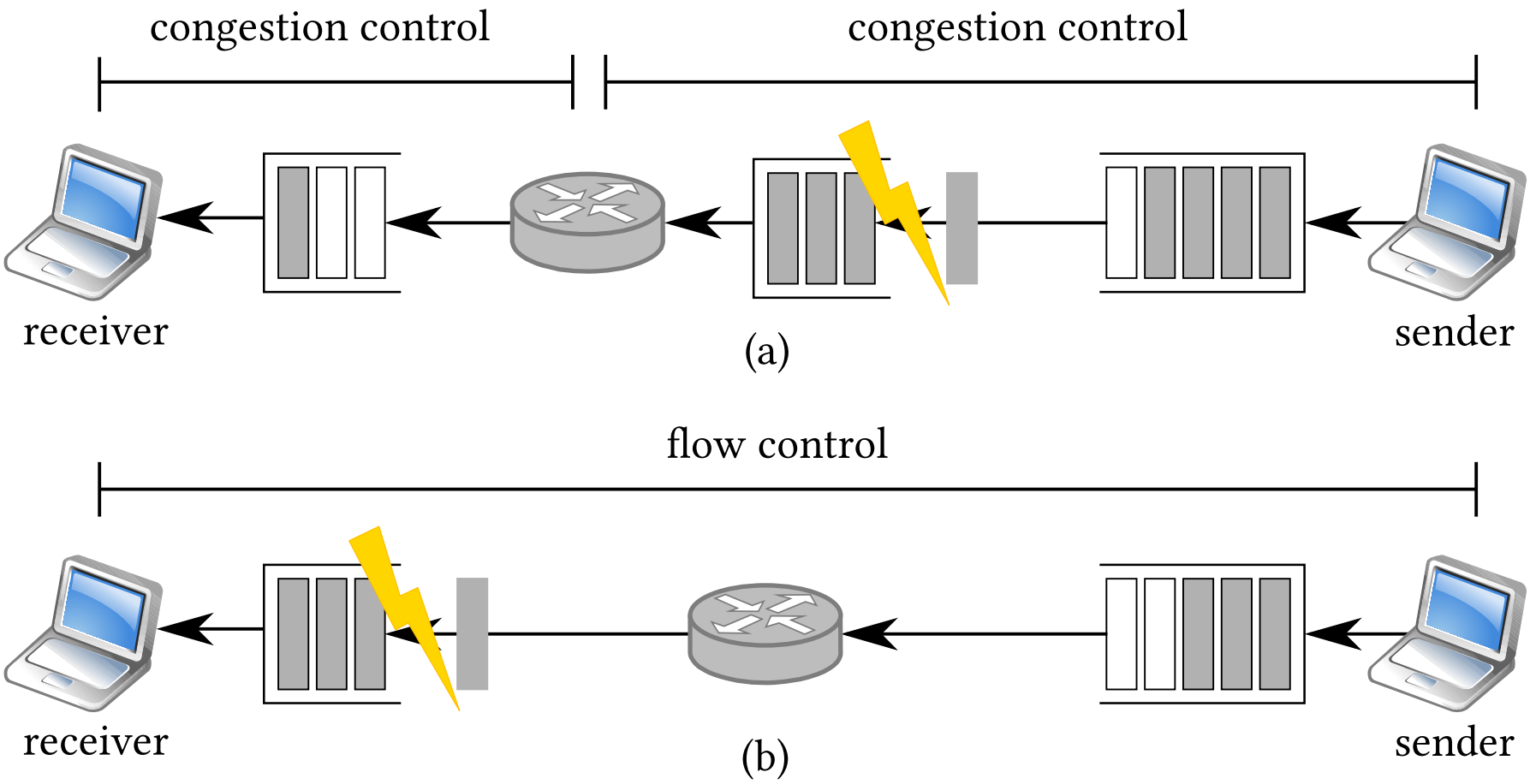
Gross Bootstrap Performance



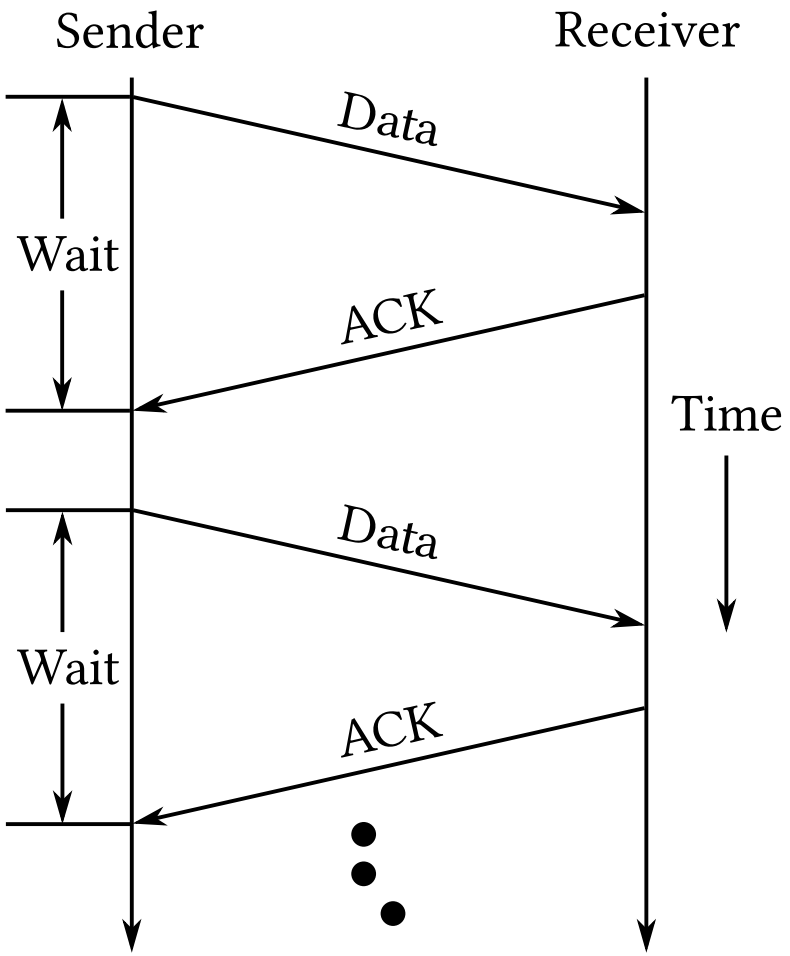
DHT Performance



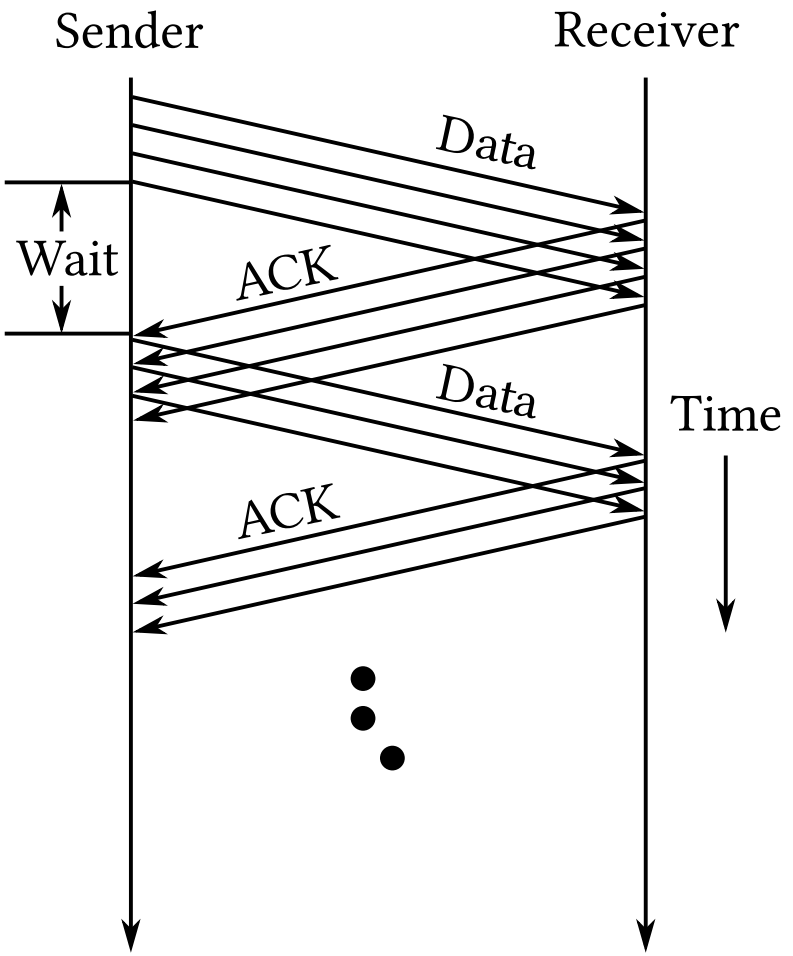
Flow Control



Flow Control



(a)



(b)

Flow Control