Technische Universität Berlin

Advanced Web Technologies

Multiscreen Technologies and Standards

Dr. Louay Bassbouss | Open Distributed Systems | lecture winter term 2023/24

Schedule

No	Week	Date (14:00- 16:00)	Topic
1	42	16.10.2023	Introduction and Framework
2	43	23.10.2023	Web Technologies Basics Media Entertainment for the Web
3	44	30.10.2023	Foundations of Media Streaming
4	45	06.11.2023	Advanced Media Streaming
5	46	13.11.2023	Multiscreen Technologies and Standards
6	47	20.11.2023	Context-Aware Media Streaming & Encoding
7	48	27.11.2023	Dynamic Advertisement
8	49	04.12.2023	Media Players - dash.js, Exoplayer
9	50	11.12.2023	HbbTV and Smart TV
	51	18.12.2023	Holiday break
	52	25.12.2023	Holiday break
	1	01.01.2024	Holiday break
10	2	08.01.2024	Media Delivery in 5G Networks (1)
11	3	15.01.2024	Media Delivery in 5G Networks (2)
12	4	22.01.2024	Metaverse Platforms and Technologies
13	5	29.01.2024	Securing Content-Provenance and Authenticity
14	6	05.02.2024	Interoperable Web-supported Learning Technologies
15	7	12.02.2024	Exercise and Test Preparation
16	8	19.02.2024	Written Test (60min) first slot (details will be announced during the semester)

TYPICAL TYPES OF SCREENS PEOPLE USE ON A DAILY BASIS

Laptop/PC

Game Console

TV/Set-Top-Boxes

Smartphones

Tablets

BUT THERE ARE MORE …

Public Screens/ Digital Signage

Smart Watches

In-car infotainment

Streaming Devices

AI Pin (Humane)

Smart Glasses

TERMINOLOGY: DEFINITIONS VS. BUZZ WORDS

There are different buzz words, but no unique definition

- **Cross-screen**: Access information or services from different screens with display optimizations and with no mutual dependency or common execution context between application instances. Instances substitute each other.

- **Multiscreen**: Participation within a common execution context involving more than one screens with application instances interacting and complementing each other.

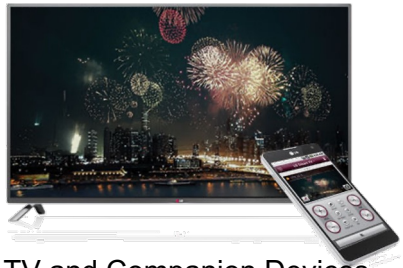
- **Second screen** (subset of Multiscreen): For the supplementary access to information and services through an automated or manual contextual link with the primary content distribution channel.

- **Companion Screen**: This term is usually used for devices that run apps linked to a given TV show or TV service.

- **Dual Screen**: another buzz word, mostly used iOS/Apple TV context

Multiscreen in Action – practical examples

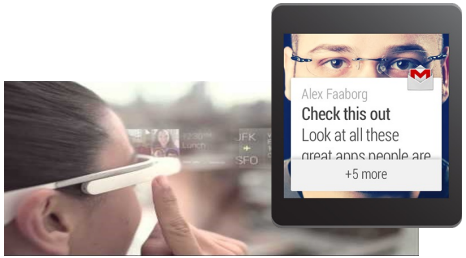
MULTISCREEN IN ACTION



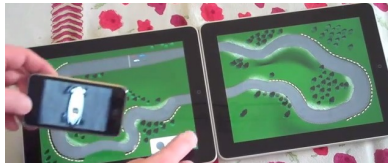
TV and Companion Devices

Streaming Devices

In-Vehicle Infotainment



Wearables

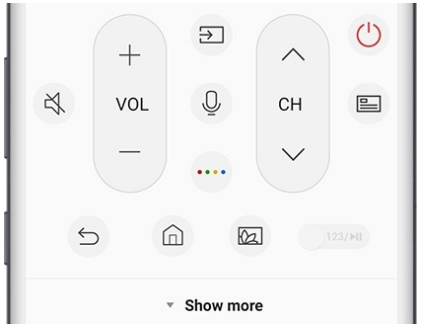


Companion devices

PC and companion devices

SAMSUNG SMART TV REMOTE CONTROL APP –SMARTTHINGS

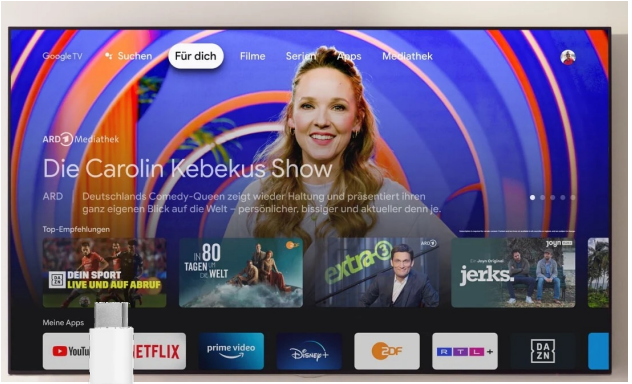
- **Remote Control**: Control TV and registered devices(STB, BDP, HTS)
- **Qwerty Keyboard** to type on TV
- **Screen Mirroring & Cast**: via SmartView

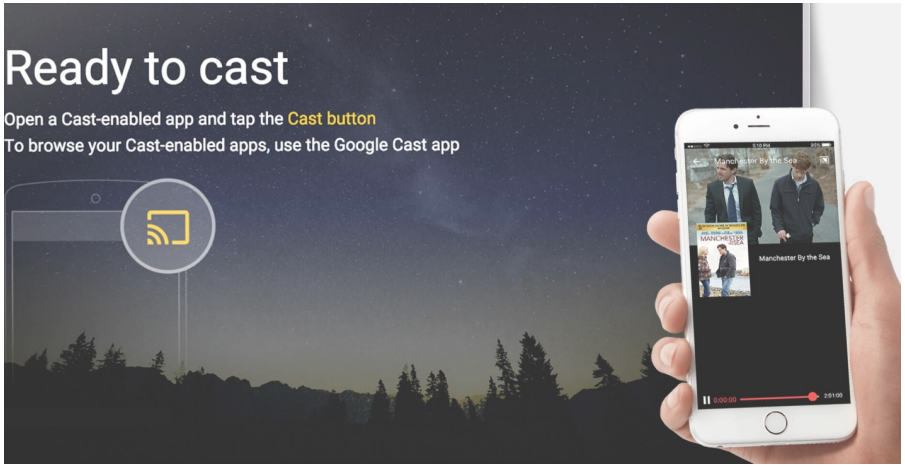


APPLE TV 4K

- Remote with voice control via Siri
- App store. Most popular Streaming Apps available
- Streaming (Video, Music, Photos) via AirPlay
- Screen Mirroring via AirPlay
- Dual Screen for supported iOS Apps

GOOGLE CHROMECAST





- Supports Google TV and Play Store
- 4K or 1080p HDR
- Remote Control with Google Assistant
- Integration with Google Mobile App, e.g. bookmark media you search and watch later on Chromecast (or any Google TV)
- Google Cast (cast media from Smartphone)

Fraunhofer FOKUS

KODI MEDIA CENTER (FORMALLY XBMC MEDIA CENTER)

- Kodi™ (formerly known as XBMC™) is an award-winning free and open source (GPL) software media center for playing videos, music, pictures, games, and more. Kodi runs on Linux, OS X, Windows, iOS, Android and Raspberry Pi, featuring a 10-foot user interface for use with televisions and remote controls
- Support for hundreds of remote controls, CEC-compatible TVs, or one of the new Smartphone and Tablet Apps
- Kodi has **several built-in UPnP A/V features**, including the ability to receive **UPnP and DLNA content** pushed to Kodi, browse UPnP and DLNA media sources, sharing an Kodi library with other UPnP and DLNA devices, and even controlling UPnP and DLNA devices
- Airplay Video Streaming supported (Mirroring not supported)

Source: http://xbmc.org/about/

AIR DISPLAY

Turns iPad/iPhone into an extra screen for computer

Source: https://www.avatron.com/apps/air-display/

Buy the Air Display app

Do this from the device that will be your extra screen.

Get the free driver

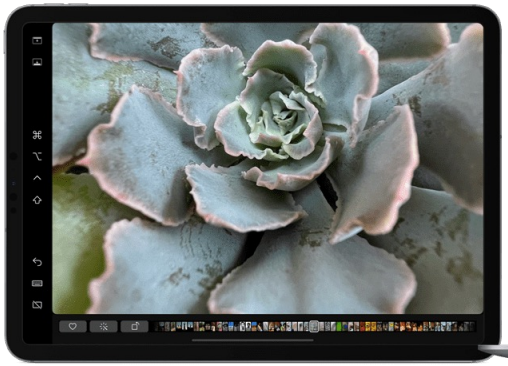
Download to the computer that will be your main screen.

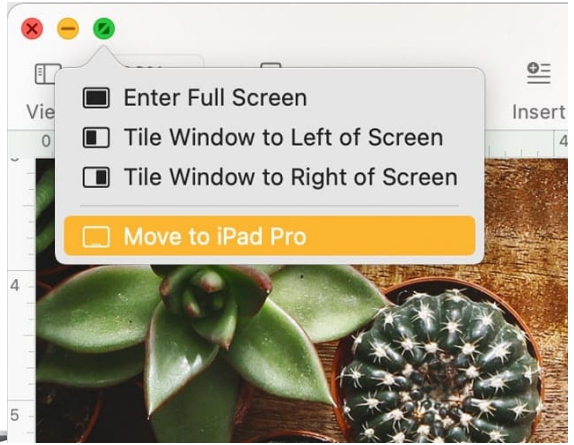
Click to connect

*Launch Air Display on your extra screen device, then **Connect** in the Air Display menu on your computer.*

APPLE SIDECAR

- With Sidecar, you can use your iPad as a display that extends or mirrors your Mac desktop
- Wireless and Wired (USB while charging) connectivity supported
- Works only with macOS
- Mirror macOS screen or extend it and move dedicated App windows from macOS to the iPad





APPLE WATCH

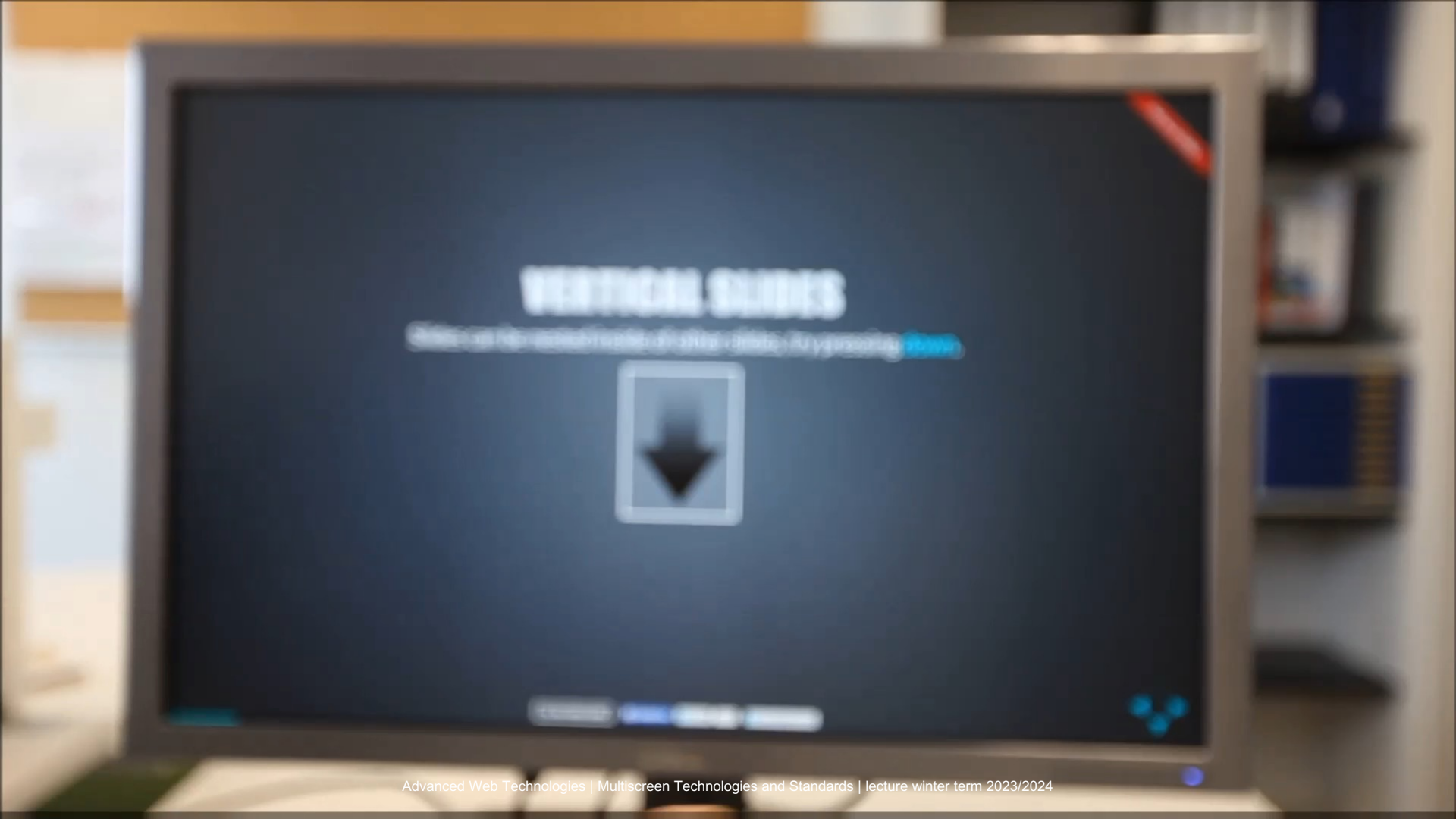
- 3 types of Extensions/Apps
  - **WatchKit Apps**: contain a Full user interface designed for Apple Watch
  - **Glances**: provide users with timely read-only information
  - **Actionable Notifications**: For Notifications and simple actions e.g. Turn Light ON

- WatchKit Apps have two parts: A WatchKit extension that runs on iPhone in the background and set of user interface resources that are installed on Apple Watch.

Source: https://developer.apple.com/watchkit/

ANDROID WEAR OS

- A multi-device experience
- Remote control phone media
- Notifications
- Messages/ Phone
- Fitness
- Music
- Google Home Control
- Nvogation


VERTICAL SLIDES

APPLE CARPLAY

Source: http://www.apple.com/ios/carplay/

APPLE CARPLAY

Source: http://www.apple.com/ios/carplay/

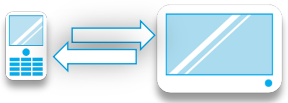
What we need to do “multiscreening”?
Common features and Challenges

TYPICAL FUNCTIONAL BLOCKS

Discovery & screen linkage

Communication & synchronization

Content adaptation & media sharing



COMMON CHALLENGES

1. **How to find screens?**

2. **How to pair screens?**

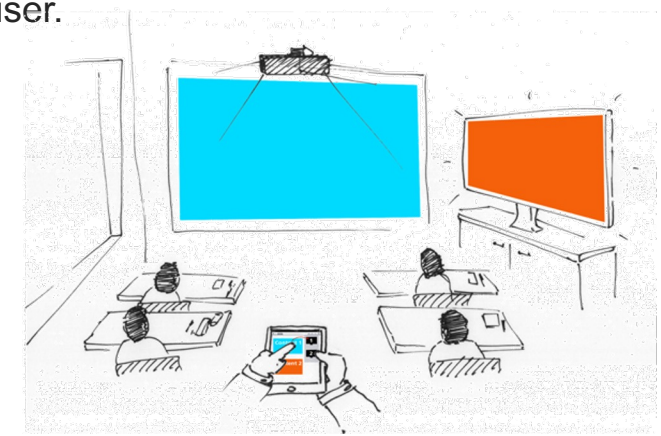
3. **How to use screens?**
   - How to share content?
   - How to launch app?
   - How to communicate?

4. **How to keep content on screens in sync?**

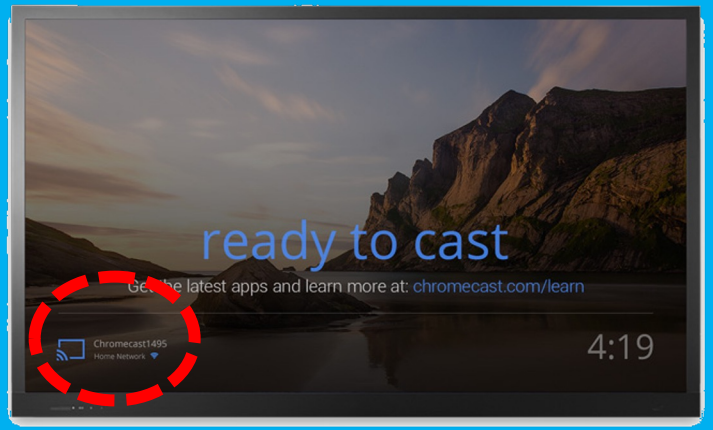
5. **How to adapt to different screens?**

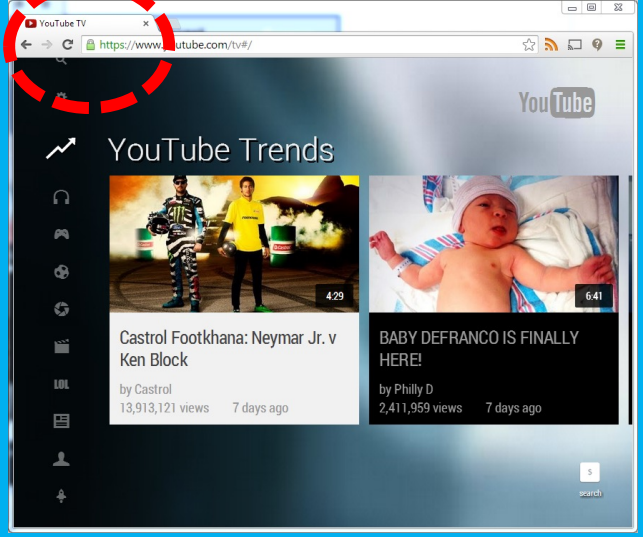
HOW TO FIND SCREENS?

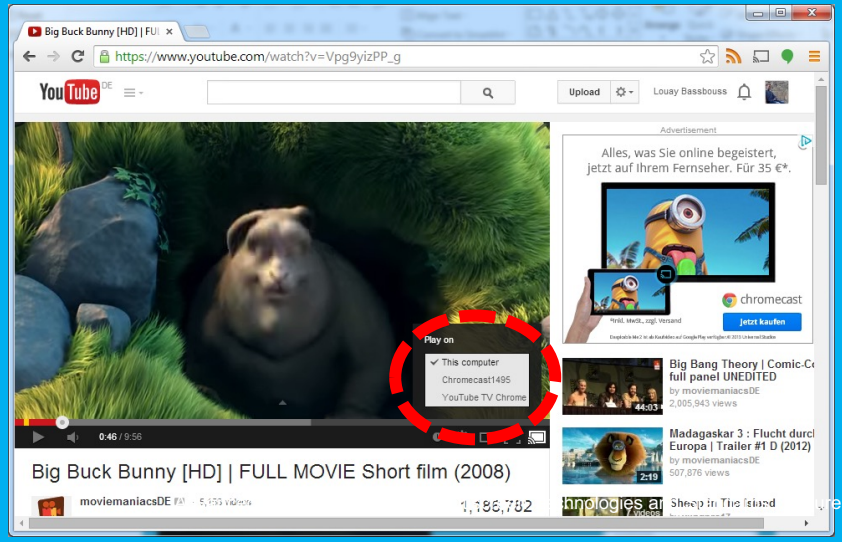
- This is this first and most important step for developing multiscreen applications → **Discovery**
- Objective of this challenge is to find one or more screens that fulfill some criteria or offer a specific service such as Video Player, Photo Viewer, etc.
- There are two basic concepts for discovering screens (or devices in general)
  - **Local Discovery**: discover devices/services nearby or in same local network (Network Service Discovery). Devices need to support human readable names.
  - **Remote Discovery**: Search devices that are registered on a central Server (Registry) and fulfil a specific criteria e.g. get all device belong to the same user.
  - Mix of Local and Remote Discovery is possible
    → YouTube
- In both cases the requesting device gets a list of available screens including names and how to connect to each of them.

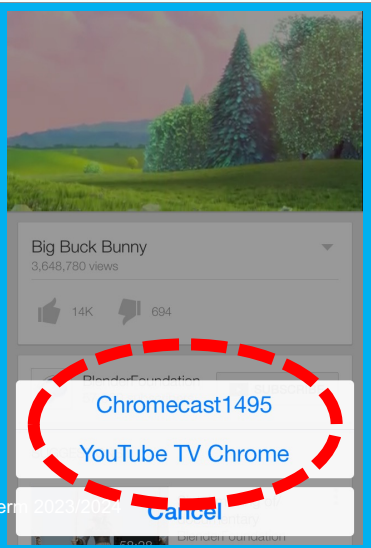


Demo


ready to cast
Get the latest apps and learn more at: chromecast.com/learn
Chromecast1495
Home Network
4:19

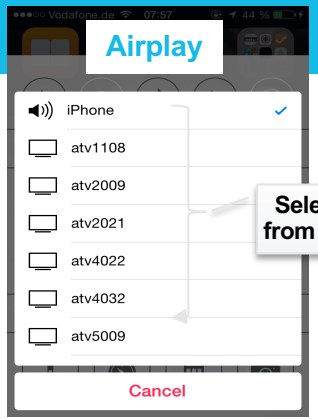

YouTube TV
YouTube Trends
4:29
Castrol Footkhana: Neymar Jr. v Ken Block
by Castrol
13,913,121 views 7 days ago
6:41
BABY DEFRANCO IS FINALLY HERE!
by Philly D
2,411,959 views 7 days ago

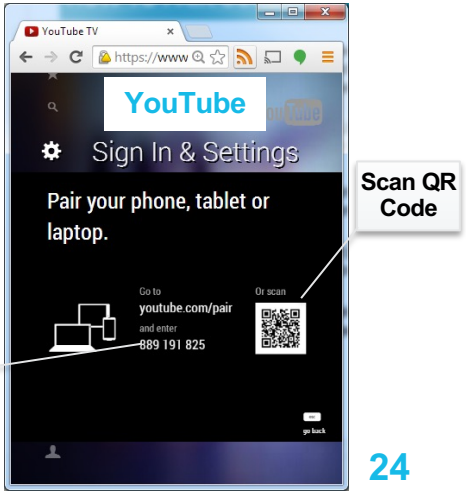

Big Buck Bunny [HD] | FULL MOVIE Short film (2008)
Play on
This computer
Chromecast1495
YouTube TV Chrome
Alles, was Sie online begeistert, jetzt auf Ihrem Fernseher. Für 35 €*.
chromecast
Jetzt kaufen
Big Bang Theory | Comic-Co full panel UNEDITED
Madagaskar 3 : Flucht durch Europa | Trailer #1 D (2012)


Big Buck Bunny
3,648,780 views
14K
694
Chromecast1495
YouTube TV Chrome
Cancel

HOW TO PAIR SCREENS?

- After the discovery process is done, the user gets a list of available screens including human readable names and connection info and other additional (optional) meta data (Location, Capabilities, etc.)
- User selects one screen from the list → User device and selected screen will be coupled with each other → **Pairing**
- There are different forms of pairing:
  - **Pairing on platform Level**: The pairing/connection is shared with all application running on top of the platform → **Airplay**
  - **Pairing on Application Level**: The pairing/connection is available for a specific application (or group of applications) → **YouTube App**
- Discovery is not always required for pairing devices. This is possible using technologies like **QR-code**, **Audio-code**, **PIN-code**, **Gesture**, etc.



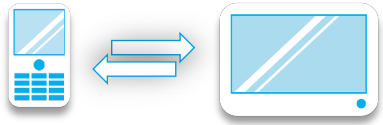


HOW TO USE SCREENS?

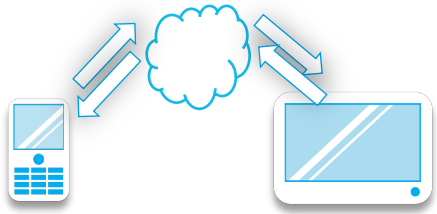
- After pairing process is done, the screens are ready for use → Different usage options:
  - **Use existing/common services/applications**: most common usage for this is **Content or Screen Sharing**. The App provider needs only to pass the content to the paired device or specific service running on that device. Video → Video Player, Music → Audio Player, Photo → Photo Viewer, etc. (technologies: DLNA, Airplay)
  - **Use specific services/applications**: In many situations, App providers need to distribute the logic of the application on the different screens. Instead of starting the application (different components) manually on each screen and then link the application components to each other, the **Remote App Launch** feature will help to do it in one step (technologies: DIAL, Chromecast).
    - A Launcher service need to be available on the paired screen. This service offers a function to Launch a specific native or Web application.
    - It should be possible to pass app specific parameters to the application during Launch. E.g.: Launch YouTube App and Play “big buck bunny” video
    - In some situations a **Wacke-up** of Application and **Notification** of user is needed before Launch. This is relevant for personal devices more than TVs for better user experience (technologies: iBeacon)

HOW TO USE SCREENS?

- In both usage options described on previous slide, a coordination between the components running on different screens is needed. The **App2App Communication** feature offers the required functionalities.
- For Content Sharing case, the communication is needed to control the media playback on the remote screen e.g. Play, Pause, Seek, etc. or for monitoring the playback state e.g. current time, etc.
- For App Launch case, the communication is needed for exchanging app
- Different Communication mechanisms:
  - **Direct Communication** (Peer2Peer): A direct communication channel between the entities will be established. This is relevant for devices in same network.



  - **Indirect Communication** (Over a Proxy or third entity): This is relevant if a direct communication is not possible. Firewall, different networks, etc.



HOW TO KEEP SCREENS IN SYNC?

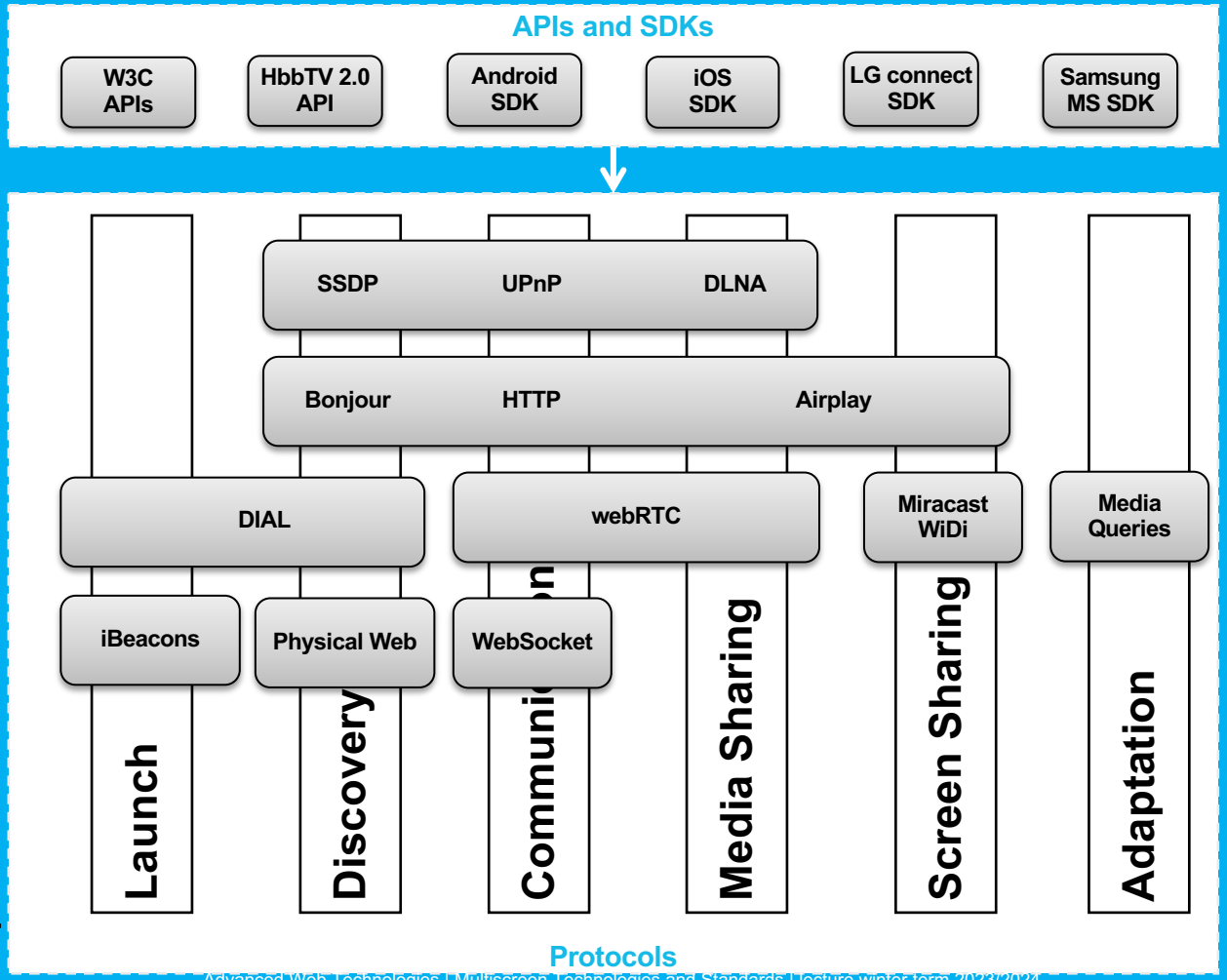
- There are different ways to do **Synchronization**:
  - **In-App Synchronization**: Uses the channels established for App2App Communication to synchronize content on the different screens
  - **Cross-App Synchronization**: The application uses other channels than for App2App Communication to synchronize content on different screens. E.g. Audio Watermarking/Fingerprinting technologies
- … and different objectives to do synchronization:
  - **Content Synchronization**: Keep Content on different screens always in sync. Updates will be immediately reflected to all connected devices (e.g. Dropbox, Google Drive, etc.)
  - **State Synchronization**: The state of the application distributed on different screens will be kept in sync during the runtime of the application (E.g. state of a multiplayer game)
  - **Timeline Synchronization**: keep the playback of different streams or time-based media in sync. This is relevant for playback of multi-stream media.

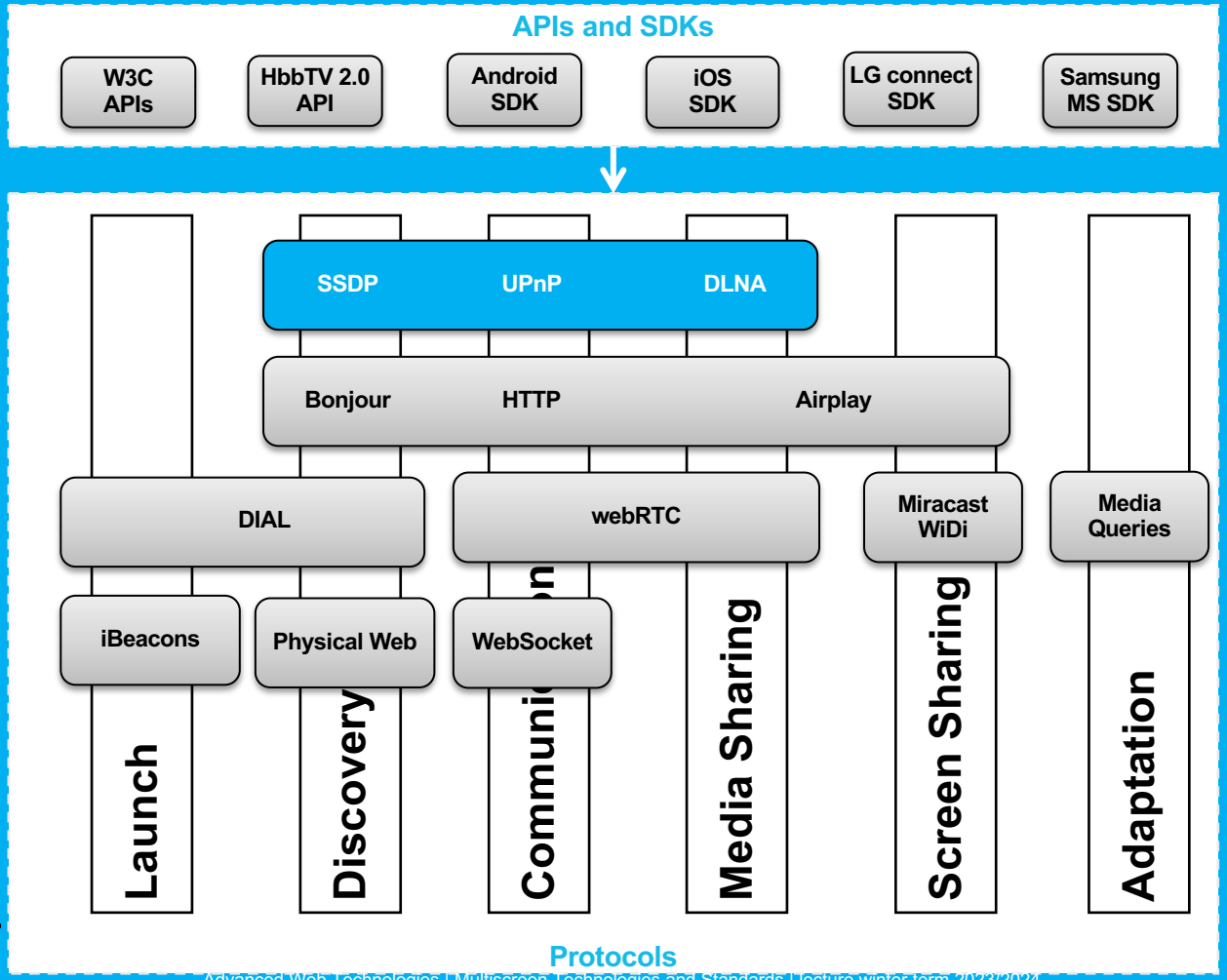
HOW TO ADAPT TO DIFFERENT SCREENS?

- Multiscreen applications run on devices with **different screen sizes/resolutions and input capabilities**.
  - Small screens on mobile devices, medium screens on PC/Laptop to large TV displays
  - Touch/Voice input on mobile, Mouse/Keyboard on PC/Laptop, Remote Control on TV
- **Adaptation** of applications to target screen is needed even on devices from same category and running same platform (iPhone 4, iPhone5)
- Different options for adaptation:
  - **Implement different apps for the different screens**: time and costs intensive
  - **Implement only UI for each screen type**: This needs to separate the UI from the Logic of the application and to define interfaces between them (MVC)
  - **Implement one App for all screens (responsive design)**: Adapt to target screen at runtime. Adaptation Technologies may help (CSS Media Queries, Twitter Bootstrap, iOS Universal Apps, …)

FEATURE-TECHNOLOGY MATRIX

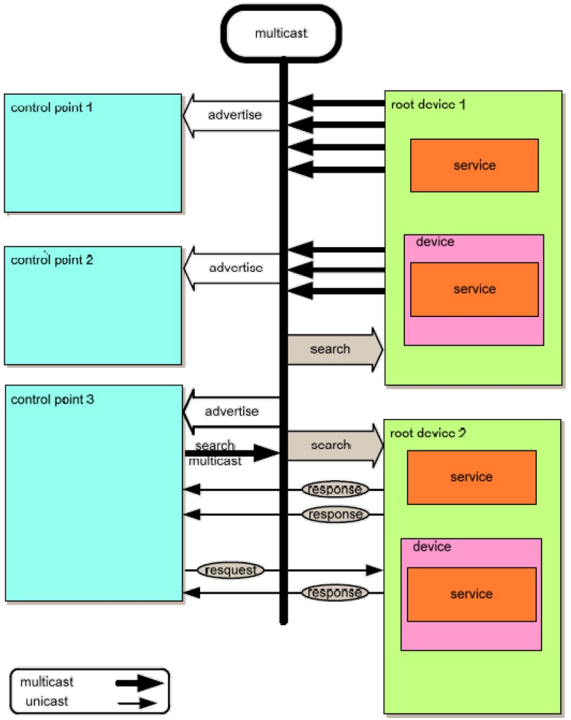
	SSDP/ UPnP/ DLNA	mDNS/ Bonjour	Miracast/ WiDi	Airplay/ AirDrop	DIAL	Audio Watermarks	QR, Pin, Audio code	WebSockets / WebRTC	W3C Presentation API	W3C Media Queries	BLE/ iBeacon	BLE/ Physical Web	HbbTV 2.0 CS
Discovery	+	+			+						+	+	+
Pairing				+			+						
Wake-up & Notification											+	+	
Launch					+								+
Communication	+							+			(+)	(+)	+
Content Sharing	+			+									
Screen Sharing			+	+					+				
Synchronization	+					+					+	+	
Adaptation										+			


APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols


APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

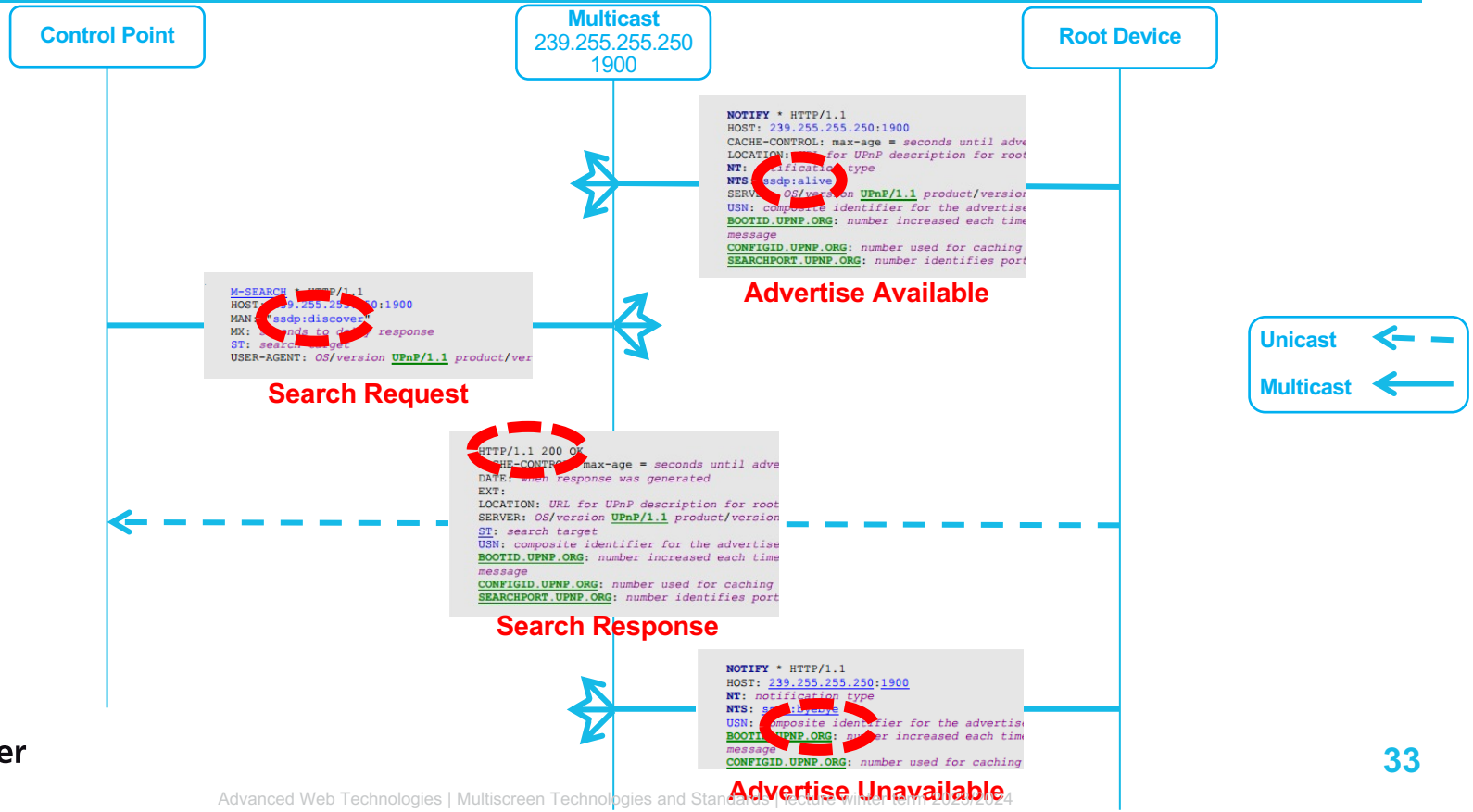
SIMPLE SERVICE DISCOVERY PROTOCOL SSDP

- **SSDP** is the **discovery** protocol for **UPnP**
- Allows devices to **advertise** their services to control points or control points to **search** for devices of interest on the network
- The fundamental exchange in both cases is a **discovery message** containing a few, essential specifics about the device or one of its services
- SSDP Message formats (**SSDP Start-line**)
  - **NOTIFY * HTTP/1.1\r\n** → **Advertisement**
  - **M-SEARCH * HTTP/1.1\r\n** → **Search**
  - **HTTP/1.1 200 OK\r\n** → **Response**
- Multicast Address and Port
  - **239.255.255.250:1900**



Source: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

SSDP MESSAGE FLOW EXAMPLE



Source: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

UNIVERSAL PLUG AND PLAY PROTOCOL UPNP

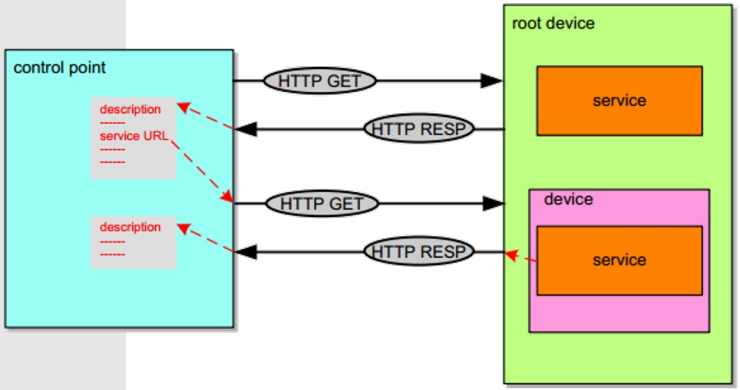
– The UPnP Device Architecture specifies **6 Steps** for UPnP Networking

Each involved device should obtain an IP Address →	(0) Addressing
UPnP Discovery is SSDP →	(1) Discovery
XML Description of Root devices and Services →	(2) Description
UPnP Device Control is based on SOAP →	(3) Control
Publish/Subscribe based, Unicast and Multicast Eventing →	(4) Eventing
Allow Root devices to provide URL for presentation in Browser →	(5) Presentation

Source: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

UPNP DEVICE DESCRIPTION EXAMPLE

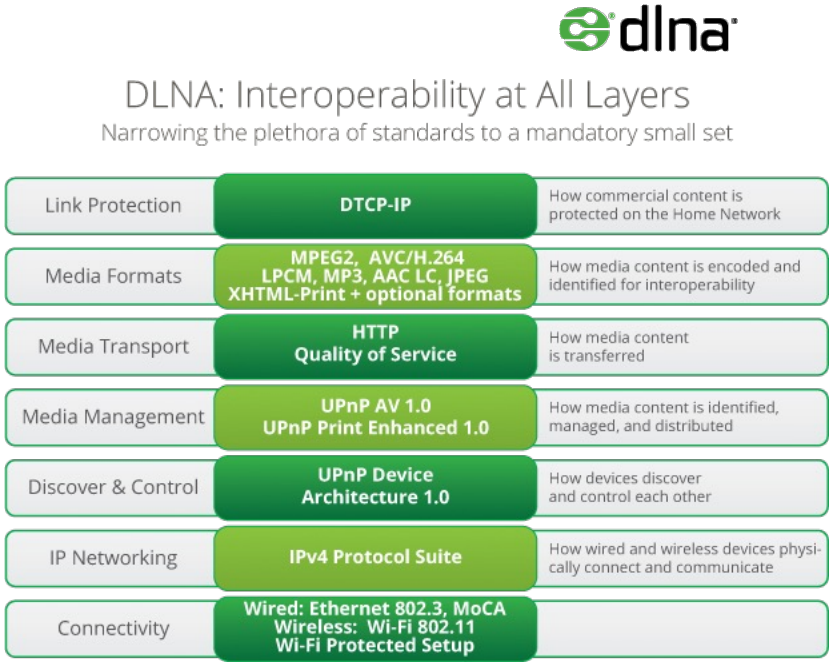
```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0"
 configId="configuration number">
    <specVersion>
        <major>1</major>
        <minor>1</minor>
    </specVersion>
    <device>
        <deviceType>urn:schemas-upnp-org:device:deviceType:v</deviceType>
        <friendlyName>short user-friendly title</friendlyName>
        <manufacturer>manufacturer name</manufacturer>
        <manufacturerURL>URL to manufacturer site</manufacturerURL>
        <modelDescription>long user-friendly title</modelDescription>
        <modelName>model name</modelName>
        <modelNumber>model number</modelNumber>
        <modelURL>URL to model site</modelURL>
        <serialNumber>manufacturer's serial number</serialNumber>
        <UDN>uuid:UUID</UDN>
        <UPC>Universal Product Code</UPC>
        <iconList>
            <icon>
                <mimetype>image/format</mimetype>
                <width>horizontal pixels</width>
                <height>vertical pixels</height>
                <depth>color depth</depth>
                <url>URL to icon</url>
            </icon>
            <!-- XML to declare other icons, if any, go here -->
        </iconList>
        <serviceList>
            <service>
                <serviceType>urn:schemas-upnp-org:service:serviceType:v</serviceType>
                <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
                <SCPDURL>URL to service description</SCPDURL>
                <controlURL>URL for control</controlURL>
                <eventSubURL>URL for eventing</eventSubURL>
            </service>
            <!-- Declarations for other services defined by a UPnP Forum working committee
                 (if any) go here -->
            <!-- Declarations for other services added by UPnP vendor (if any) go here -->
        </serviceList>
        <deviceList>
        <!-- Description of embedded devices defined by a UPnP Forum working committee
             (if any) go here -->
        <!-- Description of embedded devices added by UPnP vendor (if any) go here -->
        </deviceList>
        <presentationURL>URL for presentation</presentationURL>
    </device>
</root>
```



Source: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

DIGITAL LIVING NETWORK ALLIANCE (DLNA)

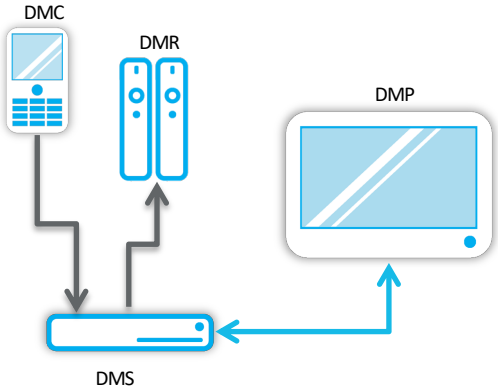
- Interoperability guidelines for sharing of digital content between devices
- Appliances, media players, consoles, computers, NAS, mobile devices, photo frames, printers, projectors, TV, STB, ...
- Over **20,000 device models** have been DLNA Certified by over **100 member companies** since the launch of the program in 2005
- **4 Billion DLNA** Certified devices



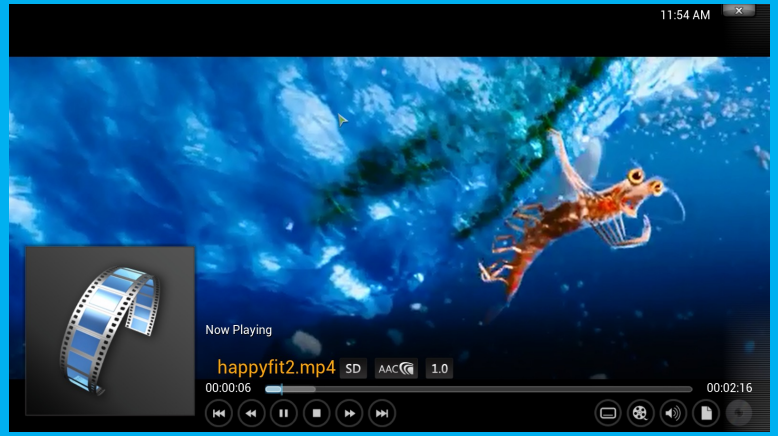
Source:http://www.dlna.org/

DLNA DEVICE TYPES FOR CONTENT SHARING

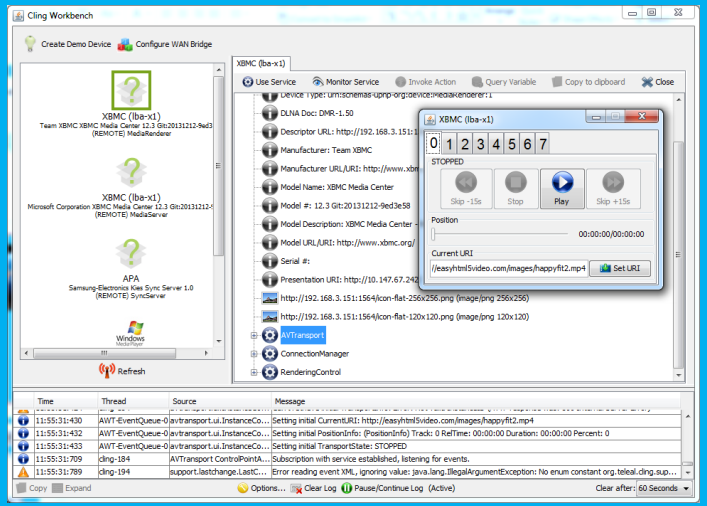
- **Digital Media Server (DMS)**
  - Provides digital content
- **Digital Media Player (DMP)**
  - Finds and plays digital content
- **Digital Media Renderer (DMR)**
  - Plays digital content
- **Digital Media Controller (DMC)**
  - Find digital content, controls playback
- **Digital Media Printer (DMPr)**
  - Network printing



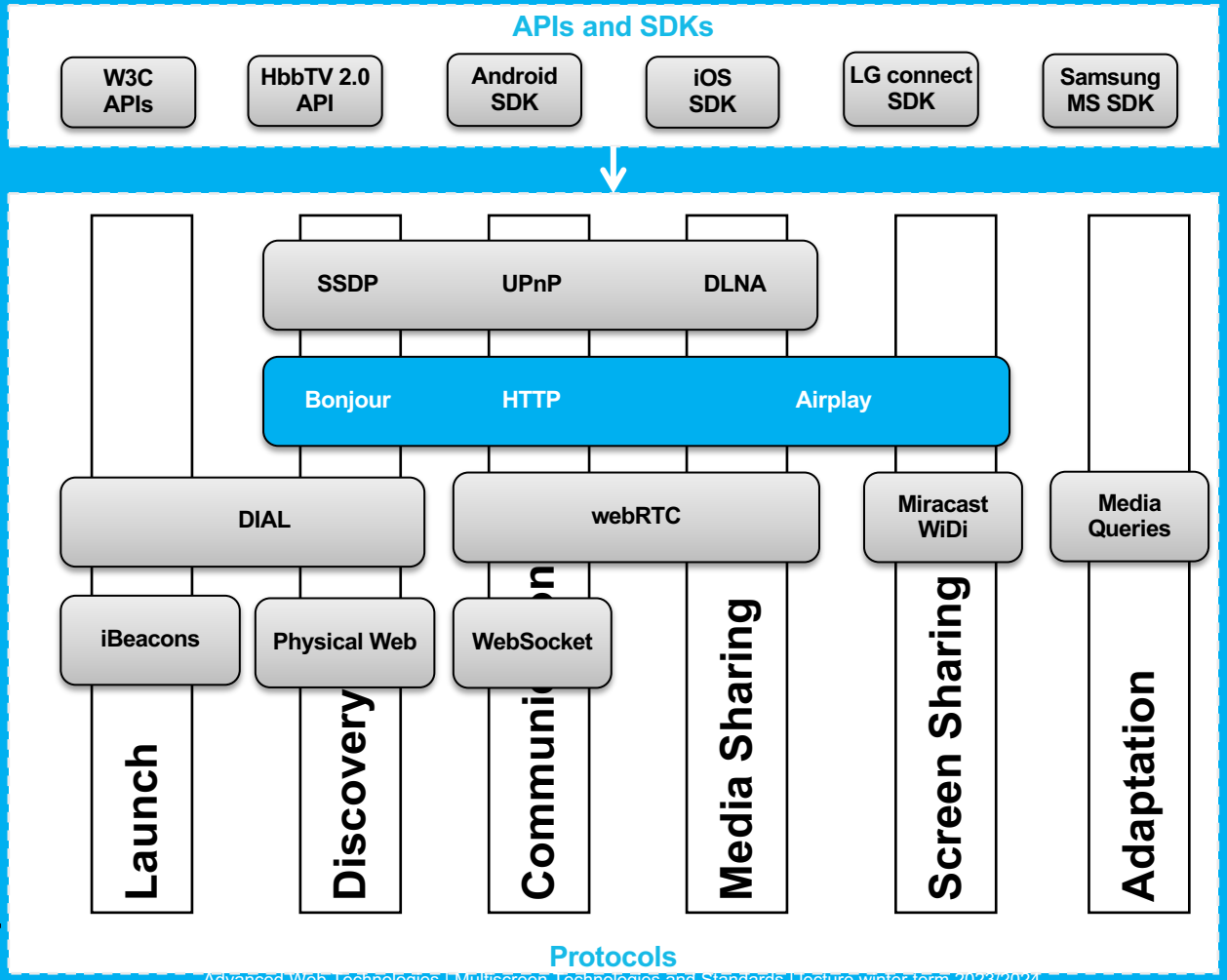
KODI UPnP Media Renderer



UPnP Demo



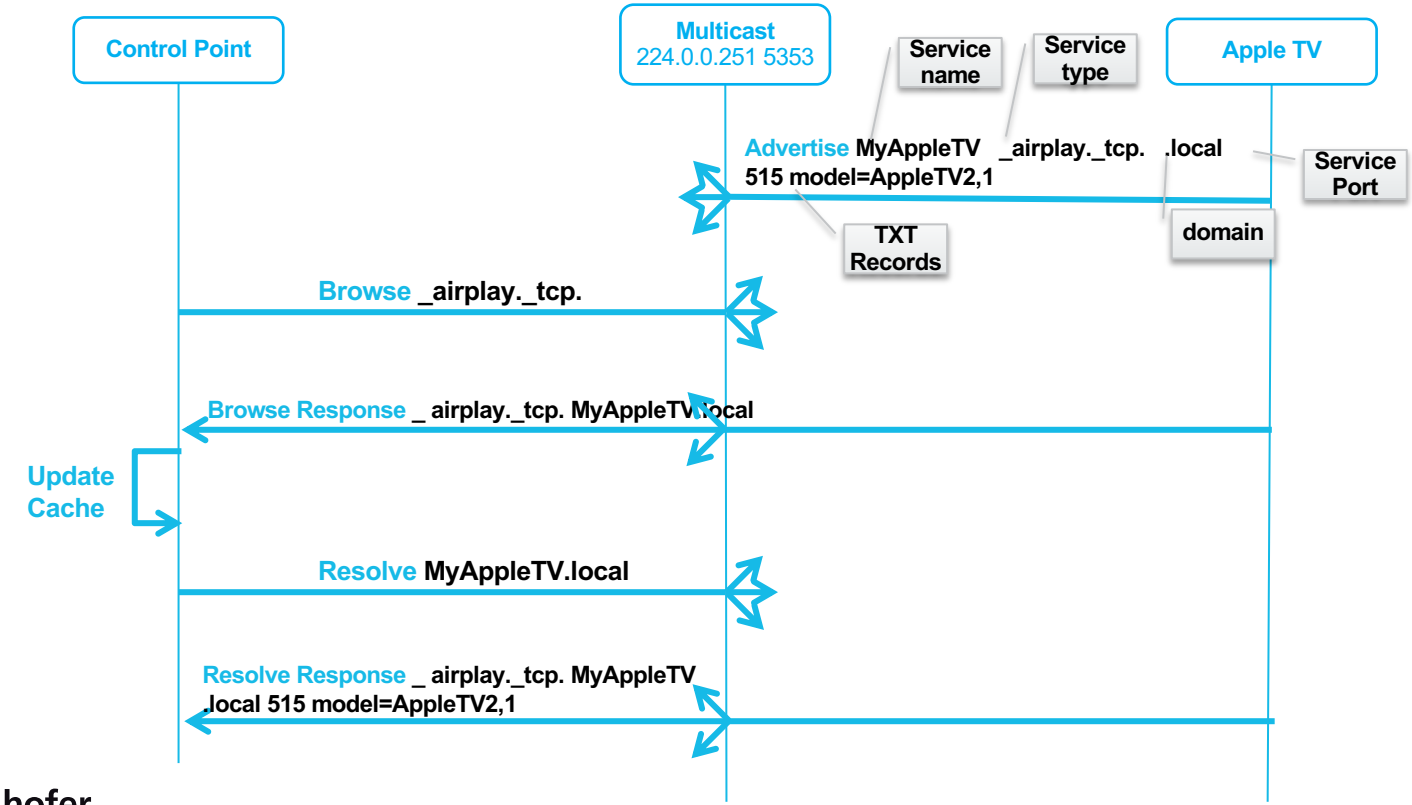
Cling UPnP Workbench


APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

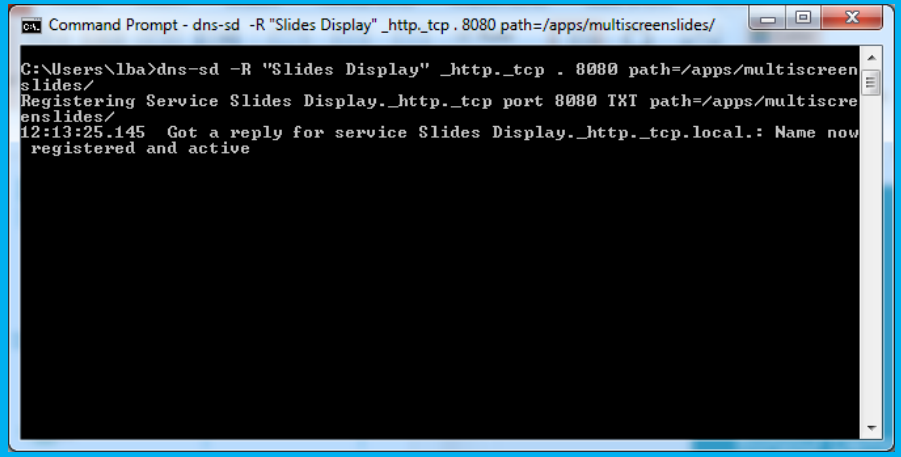
MULTICAST DNS (MDNS) - DNS SERVICE DISCOVERY (DNS-SD)

- A way of using DNS in a small local network where no conventional DNS server has been installed.
- mDNS queries are sent via **multicast** UDP to 224.0.0.251 (or FF02::FB IPv6) and port 5353
- mDNS responses are sent to all hosts in the network and update their caches
- Defines special top-level domain (TLD) ".local." which is always link to local
- Multicast requests and responses are cached to reduce network traffic
- The Service Discovery is a two-step process: (1) find the name of hosts providing a certain service and (2) resolve the IP Address, Port and other info for a specific host name.
- DNS-SD uses, PTR, SRV and DNS TXT records
- DNS Pointer records PTR define mapping between service type and list of service names e.g. *_airplay._tcp.local. 28800 PTR MyAppleTV._airplay._tcp.local.*
- DNS SRV records specifies protocol of service types e.g. *_airplay._tcp*
- DNS TXT records defines optional additional information as key-value pairs e.g. *model=AppleTV2,1 deviceid=58:55:….*

MDNS/DNS-SD MESSAGE FLOW EXAMPLE



Bonjour Demo
Browse Airplay and Chromecast Devices



dns-sd command line tool

AIRPLAY

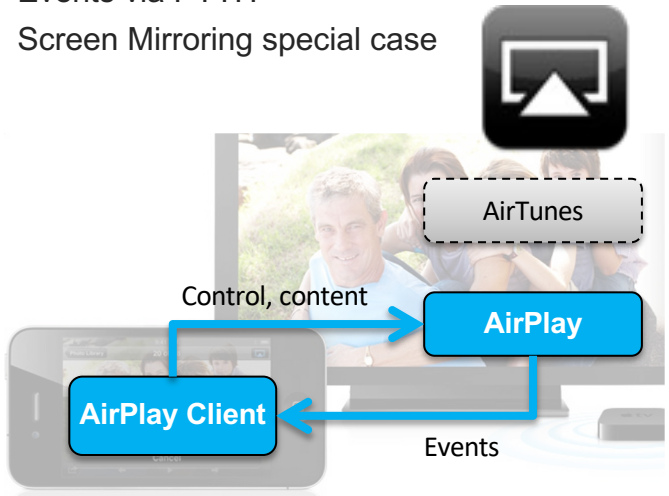
- Uses Bonjour, Apple implementation of mDNS and DNS-SD, for service discovery
- AirPlay mDNS Service type is ***_airplay._tcp***

AirPlay service

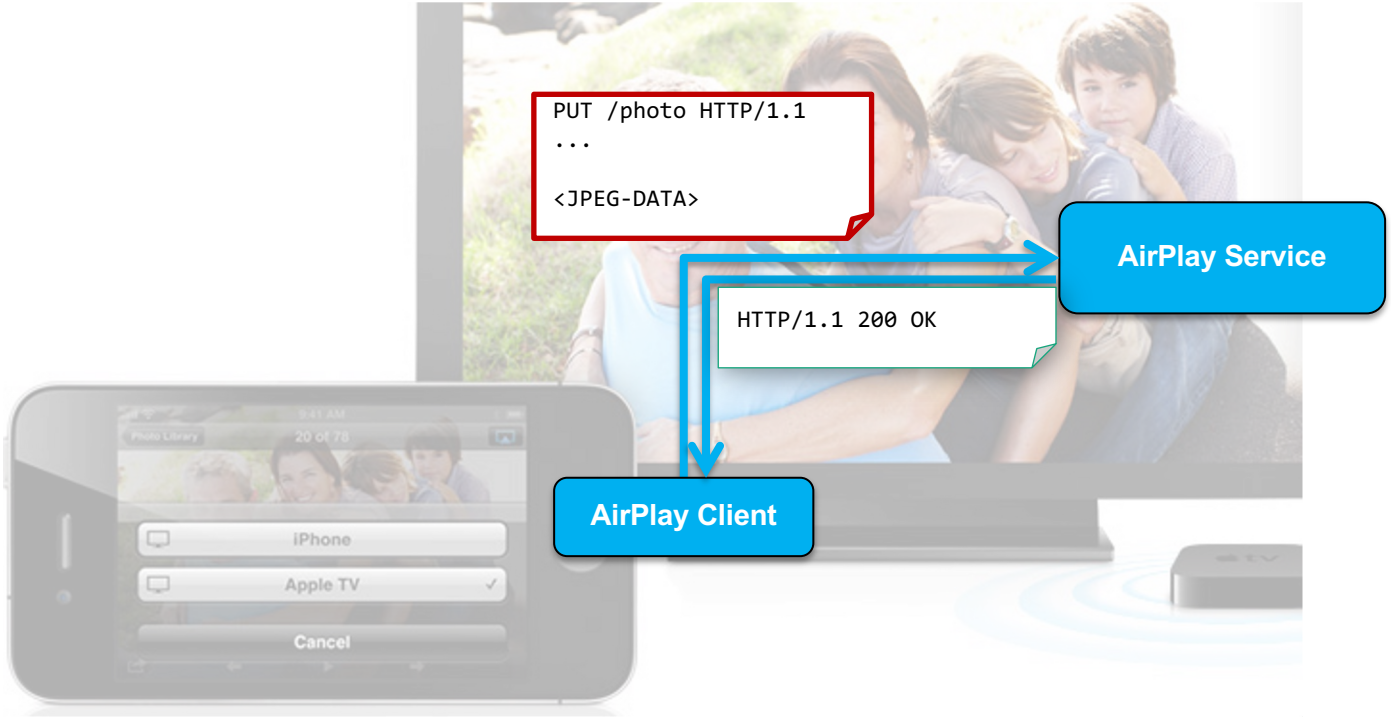
```
name: Apple TV
type: _airplay._tcp
port: 7000
txt:
 deviceid=58:55:CA:1A:E2:88
 features=0x39f7
 model=AppleTV2,1
 srcvers=130.14
```

AIRPLAY

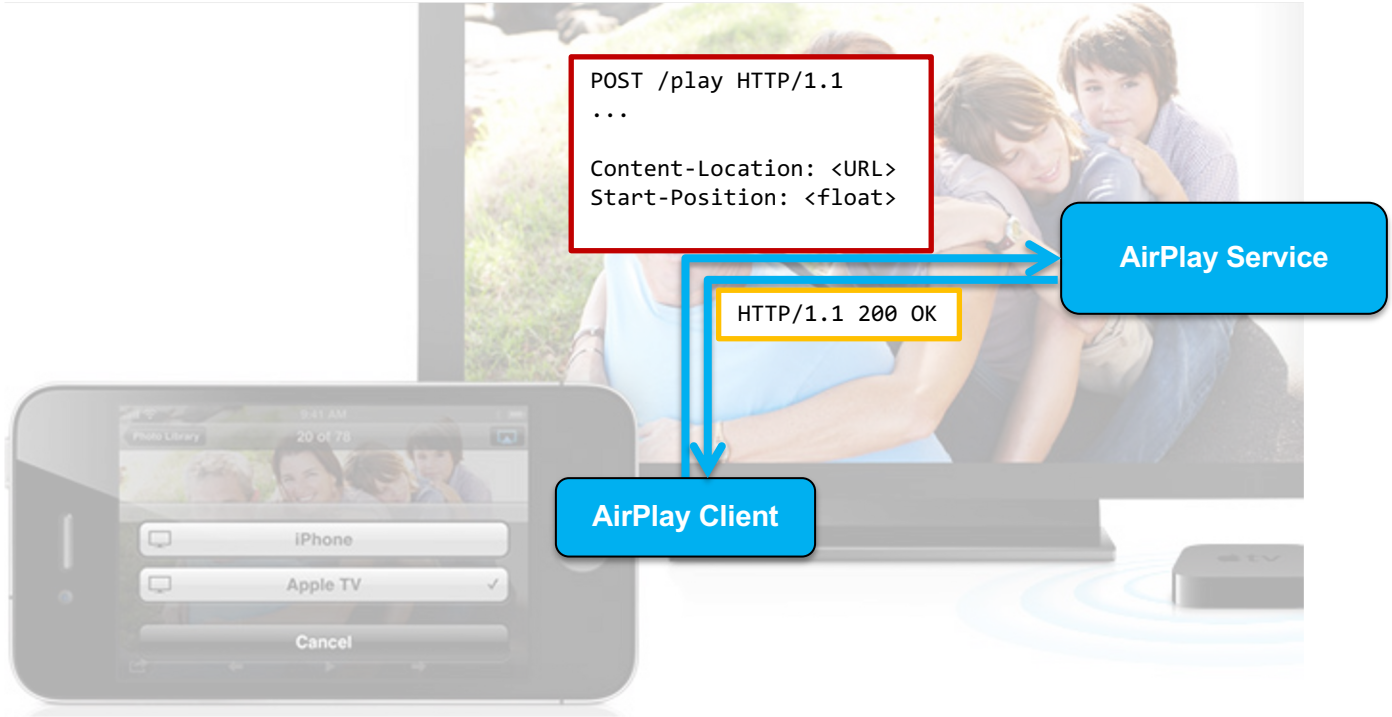
- Connect one {iPhone, iPad, MacBook, ...} to an Apple TV
  - Stream audio, photos, videos
  - Mirror screen
- Apple TV advertises two services
  - AirTunes
    - Remote Audio Output Protocol (RAOP)
    - RTSP Server
    - Audio Streaming
  - AirPlay
    - HTTP Server
    - Photo slideshows and video streaming
  - Service discovery over Bonjour (mDNS, DNS-SD)
  - Screen Mirroring special case
- Two connections between AirPlay service and client
  - Control and content via HTTP
  - Events via PTTH
  - Screen Mirroring special case



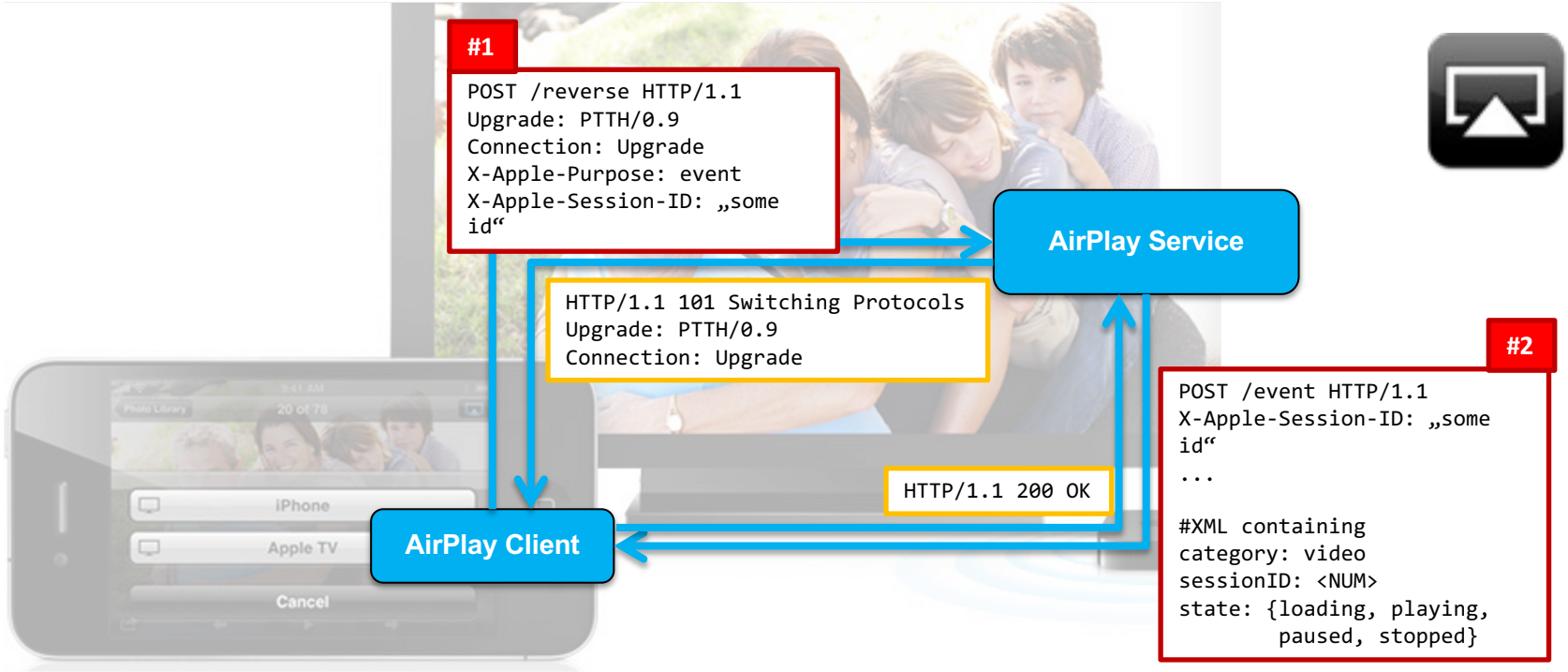
AIRPLAY: PHOTO SHARING



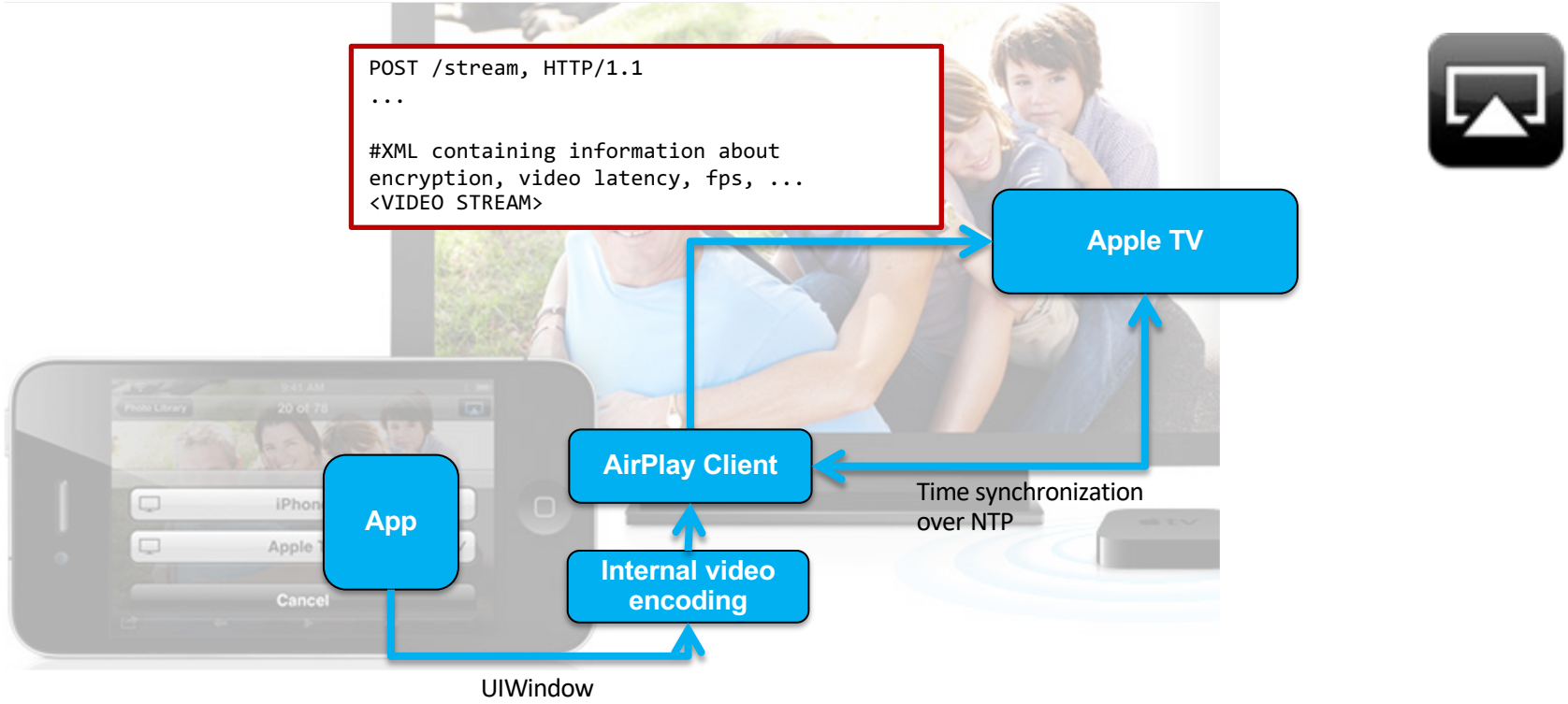
AIRPLAY: VIDEO SHARING

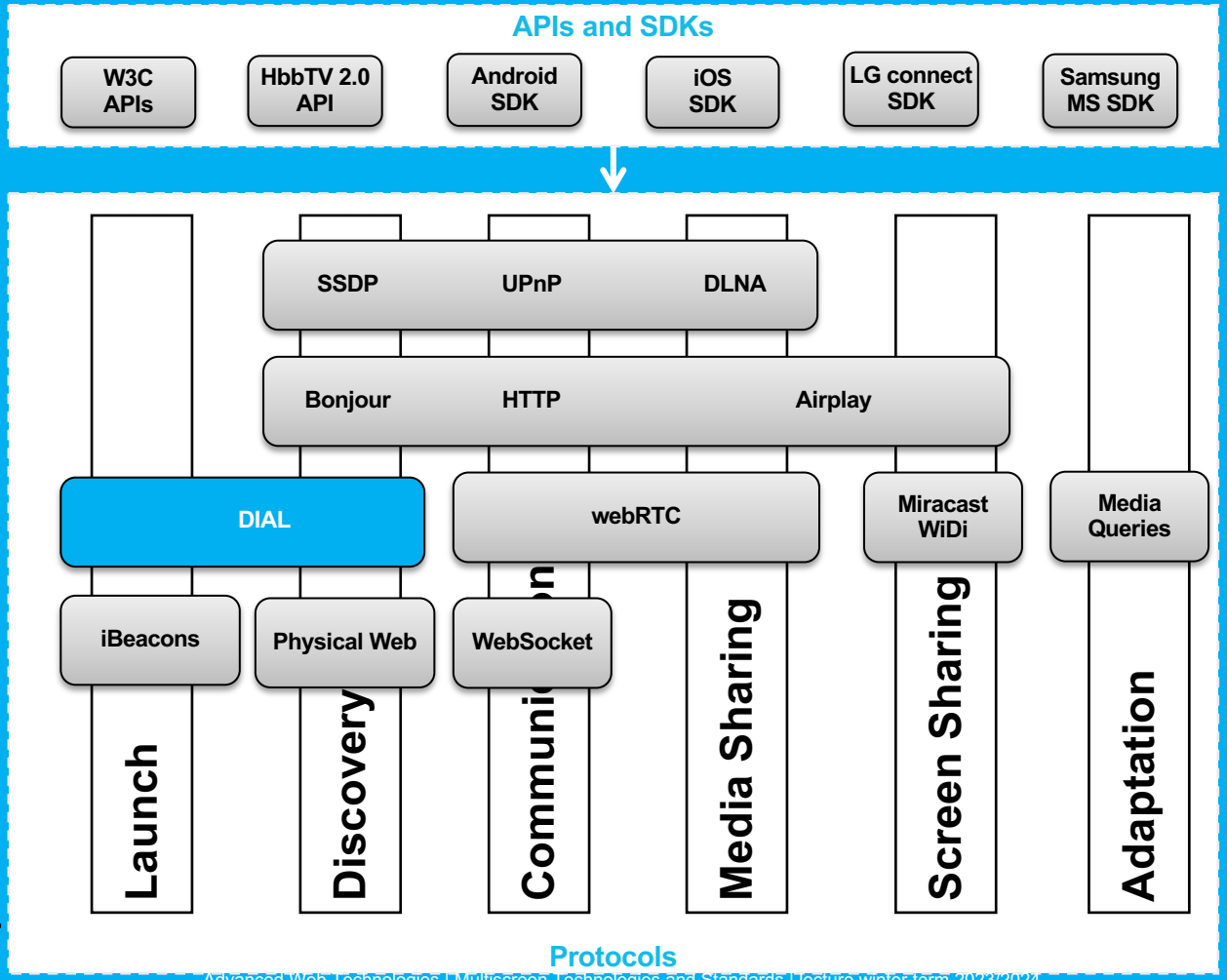


AIRPLAY: EVENTS OVER PTTH



AIRPLAY: SCREEN MIRRORING




APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

DISCOVERY AND LAUNCH PROTOCOL DIAL

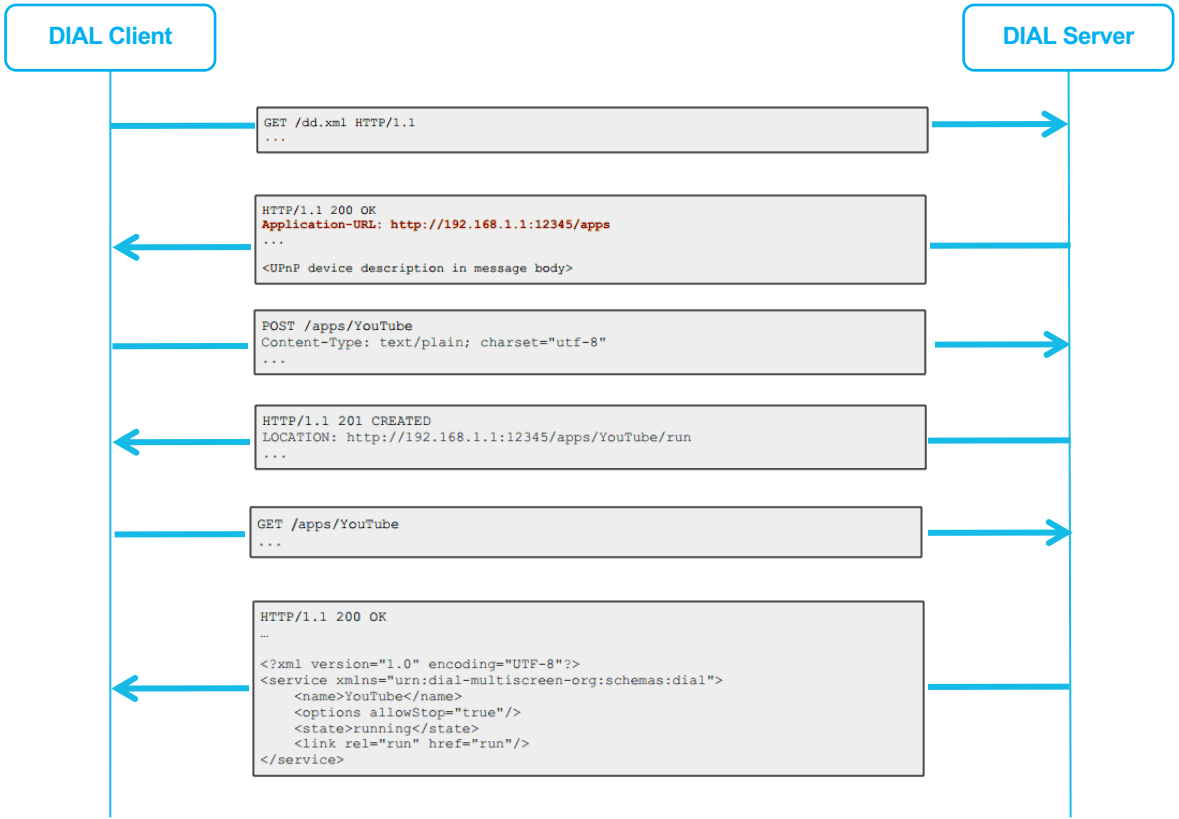
- Is a simple protocol that 2nd screen devices (Mobile devices) can use to discover and launch apps on 1st screen devices (TV and Large displays)
- Uses Discovery (SSDP) and Device/Service descriptions from UPnP Stack
- DIAL UPnP Service type is: *urn:dial-multiscreen-org:service:dial:1*
- New Layer for Launch in form of a REST API
- The end-point of the REST API is returned in the HTTP Header "*Application-URL*" of the device description response
- Use HTTP GET to check if application exists
  - GET <Application-URL>/YouTube → 200 OK response means YouTube exists
- Use HTTP POST to Launch an application
  - POST <Application-URL>/YouTube → 201 CREATED response means YouTube App is launched successfully
- Use HTTP DELETE to stop an application
  - DELETE <Application-URL>/<id> stops an Application with running instance <id>

(1) Discovery
(2) Description
(3) Launch

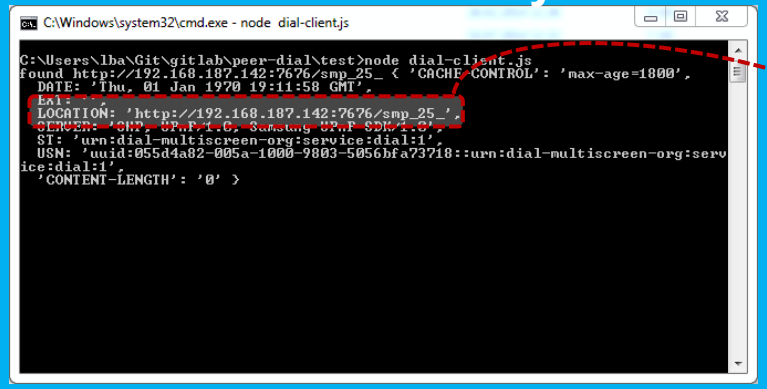
UPnP

Source: http://www.dial-multiscreen.org/

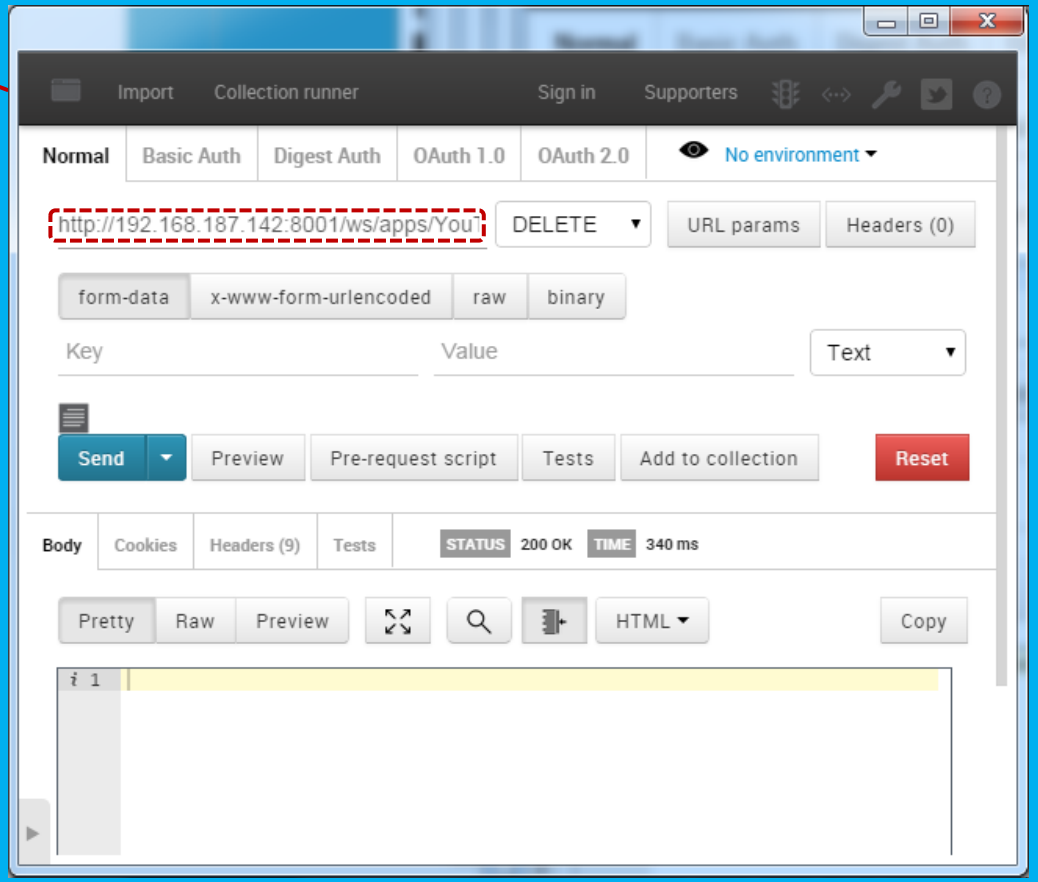
DIAL MESSAGE FLOW EXAMPLE

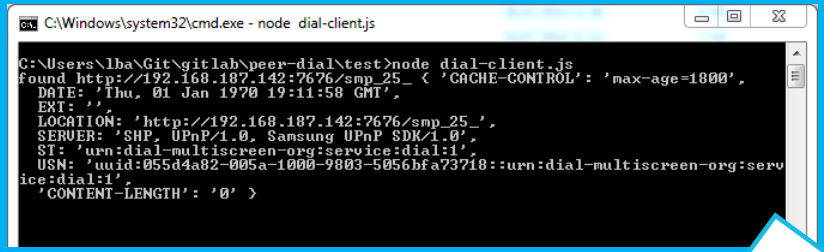


SSDP Discovery



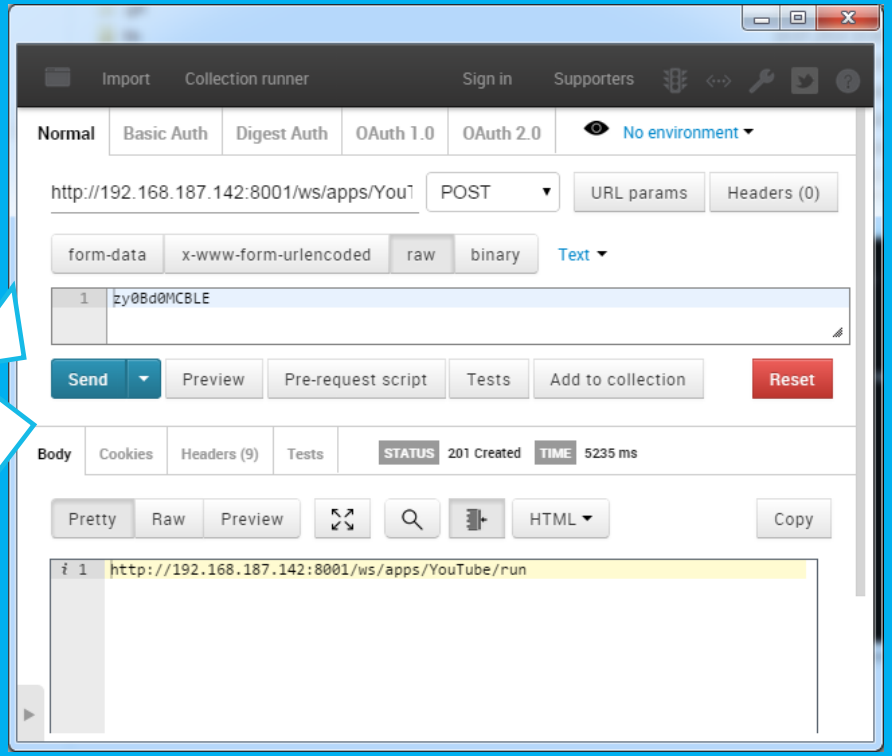
HTTP Client

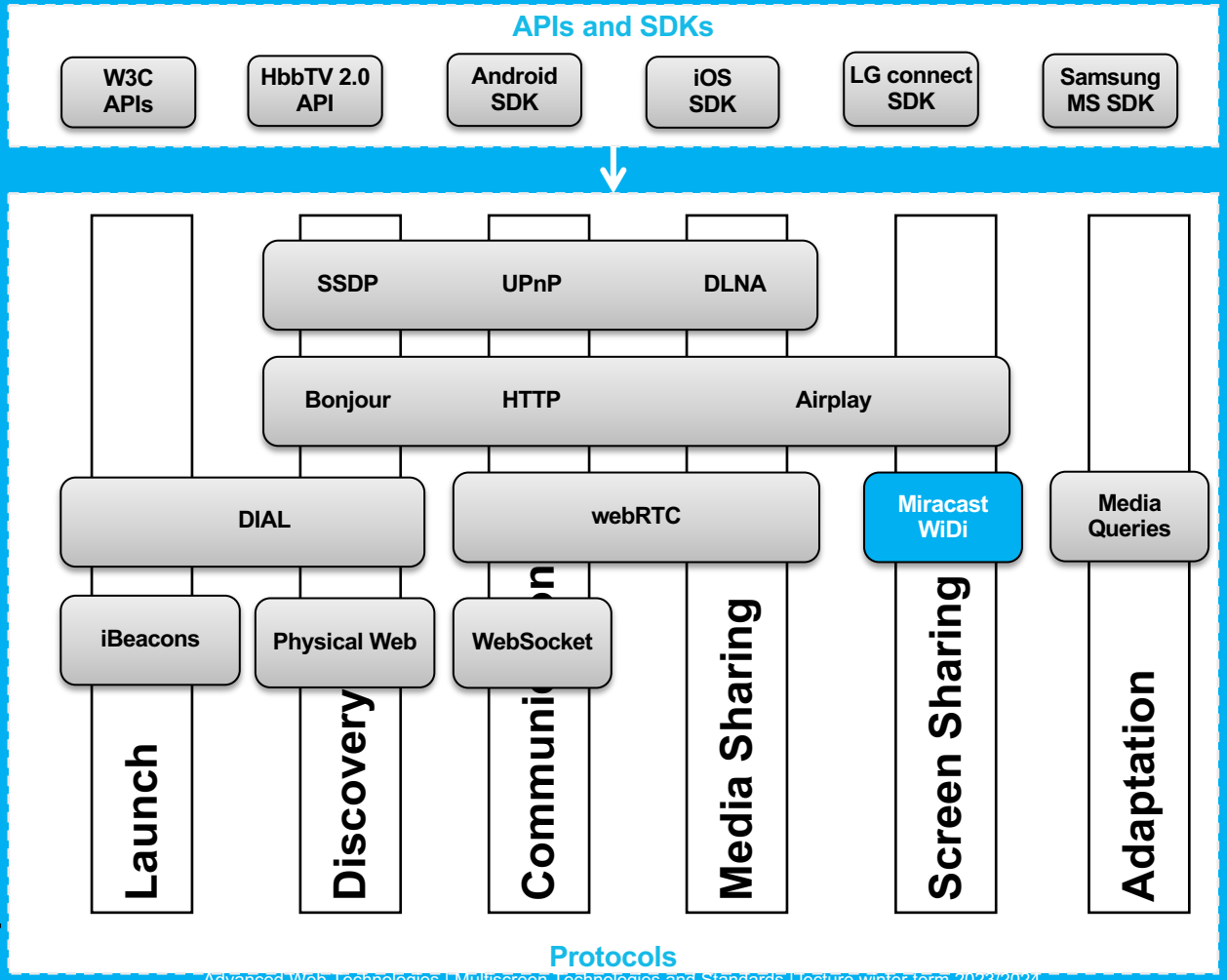



C:\Windows\system32\cmd.exe - node dial-client.js
C:\Users\lba\Git\gitlab\peer-dial\test>node dial-client.js
found http://192.168.187.142:7676/smp_25_ { 'CACHE-CONTROL': 'max-age=1800',
DATE: 'Thu, 01 Jan 1970 19:11:58 GMT',
EXT: '',
LOCATION: 'http://192.168.187.142:7676/smp_25_',
SERVER: 'SHP, UPnP/1.0, Samsung UPnP SDK/1.0',
ST: 'urn:dial-multiscreen-org:service:dial:1',
USN: 'uuid:055d4a82-005a-1000-9803-5056bfa73718::urn:dial-multiscreen-org:service:dial:1',
'CONTENT-LENGTH': '0' }

DIAL Demo


YouTube


Import
Collection runner
Sign in
Supporters
Normal
Basic Auth
Digest Auth
OAuth 1.0
OAuth 2.0
No environment
http://192.168.187.142:8001/ws/apps/YouT
POST
URL params
Headers (0)
form-data
x-www-form-urlencoded
raw
binary
Text
1
zy0Bd0MCBLE
Send
Preview
Pre-request script
Tests
Add to collection
Reset
Body
Cookies
Headers (9)
Tests
STATUS
201 Created
TIME
5235 ms
Pretty
Raw
Preview
HTML
Copy
1
http://192.168.187.142:8001/ws/apps/YouTube/run


APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

MIRACAST/INTEL WIDI

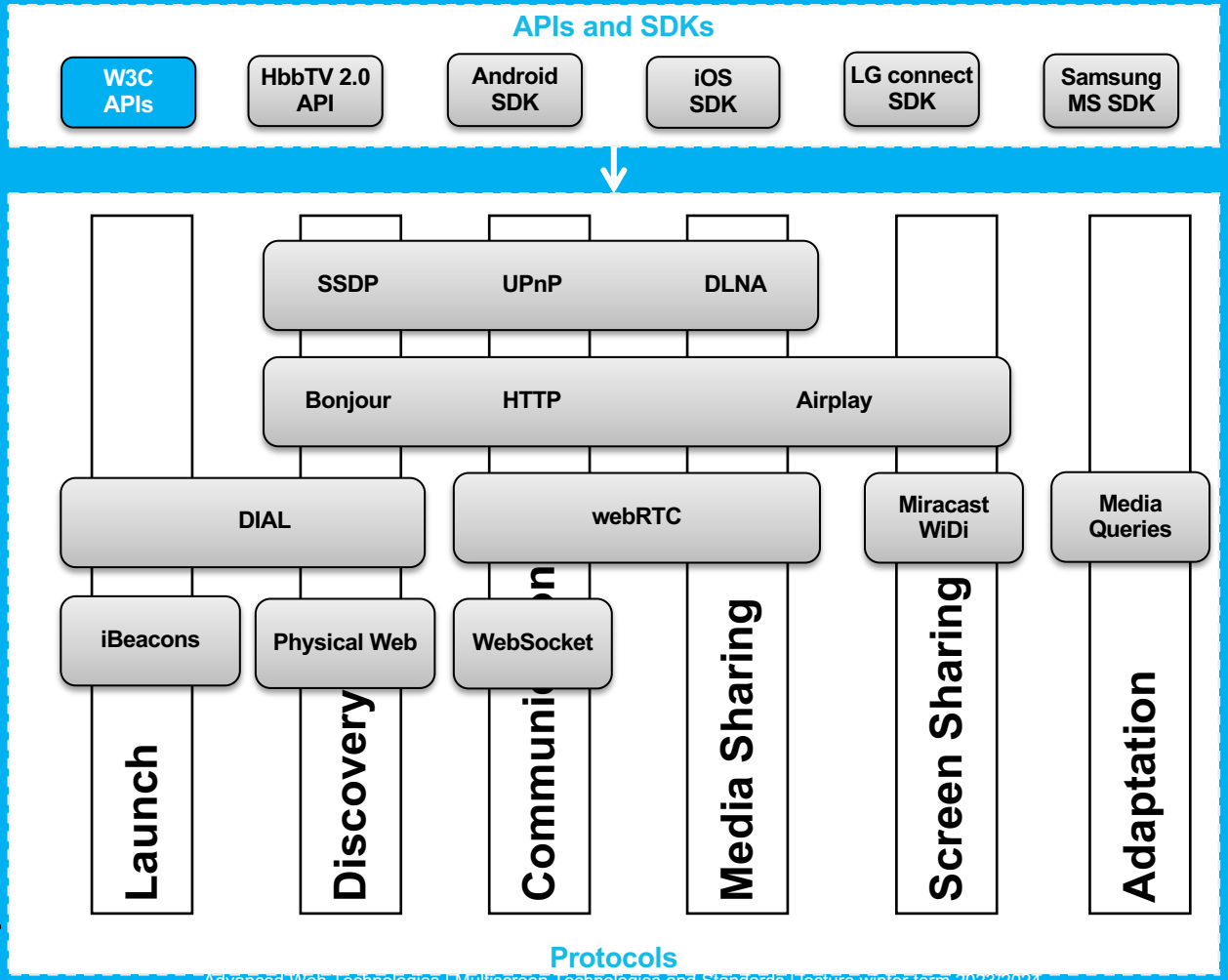
- Peer-to-peer wireless screencast standards
- Wi-Fi Direct for wireless ad-hoc peer-to-peer connections

- **Miracast**
- Developed by Wi-Fi Alliance as open platform-independent standard
- First devices certified in 09/2012
- Today 540 source devices, 1771 target devices are (2014-05-05)
- Smartphones and Wi-Fi adapters
- STB, Wi-Fi adapters, TVs, blue-ray players, ...
- LG, Samsung, Toshiba, Sony, Philips, Sharp, Intel, Qualcomm, TI, Realtek , Broadcom, NVIDIA, ...

- Support 1080p video, 5.1 audio
- **Official support in Android 4.2+**

- **Intel WiDi**
- Introduced 2010 by Intel, proprietary
- Processor renders second virtual display, broadcast via Wi-Fi
- Intel My Wi-Fi implements Wi-Fi Direct
- Supported by
- LG, Toshiba, Samsung, Sony, Belkin, Netgear
- **Since v3.5 fully compatible to Miracast**

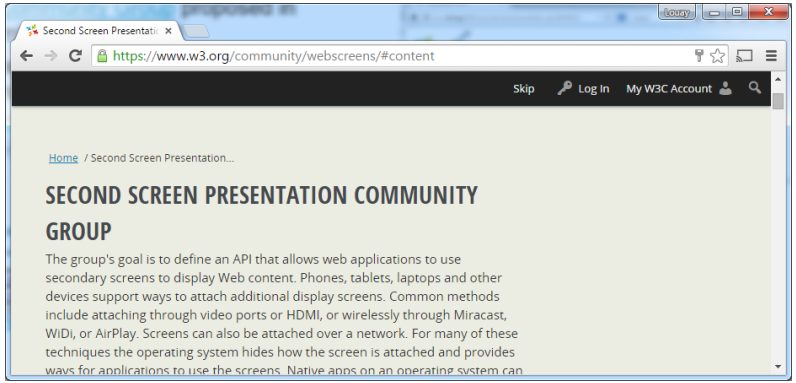
Source: http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast

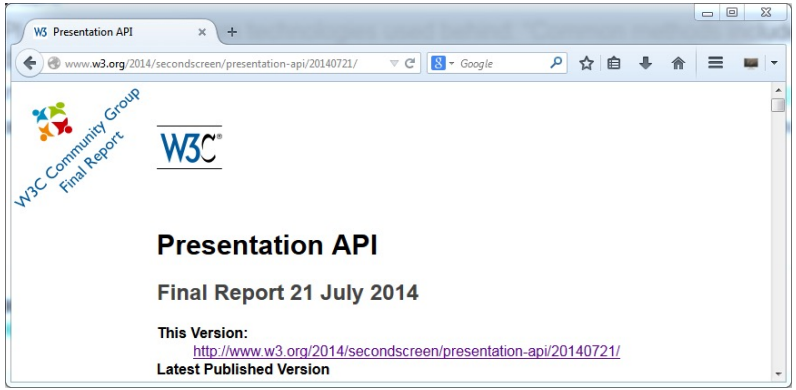

APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

W3C Second Screen Presentation API

W3C SECOND SCREEN PRESENTATION CG

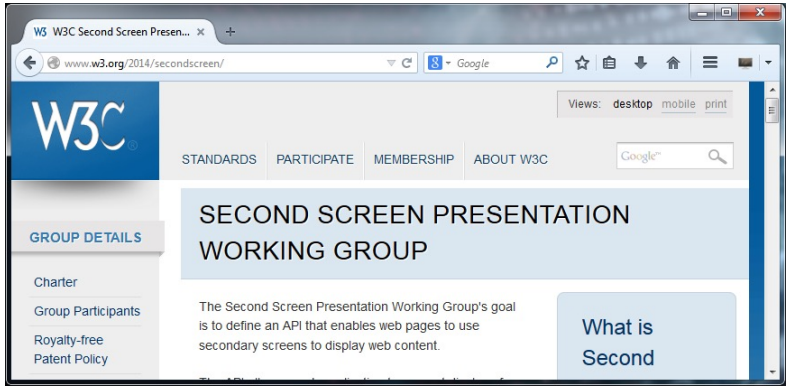
- W3C Community Group proposed in September 2013 by Intel
- Key partners: Intel, Google, Mozilla, Fraunhofer FOKUS, Netflix, LGE, etc.
- Goal: “Is to define an API that allows web applications to use secondary screens to display Web content”
- Final Report of the CG published in July 2014.

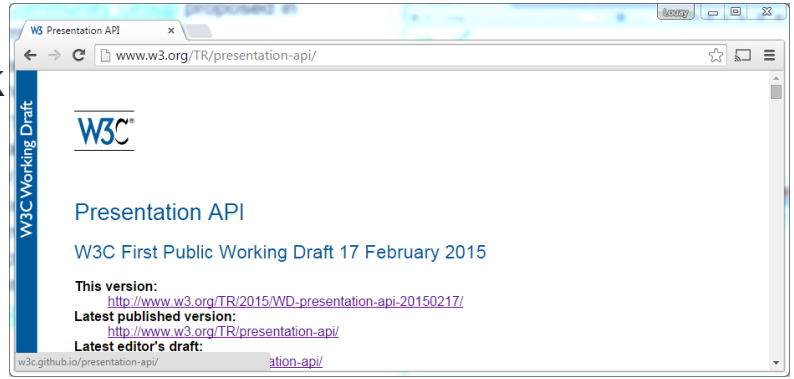




W3C SECOND SCREEN WG

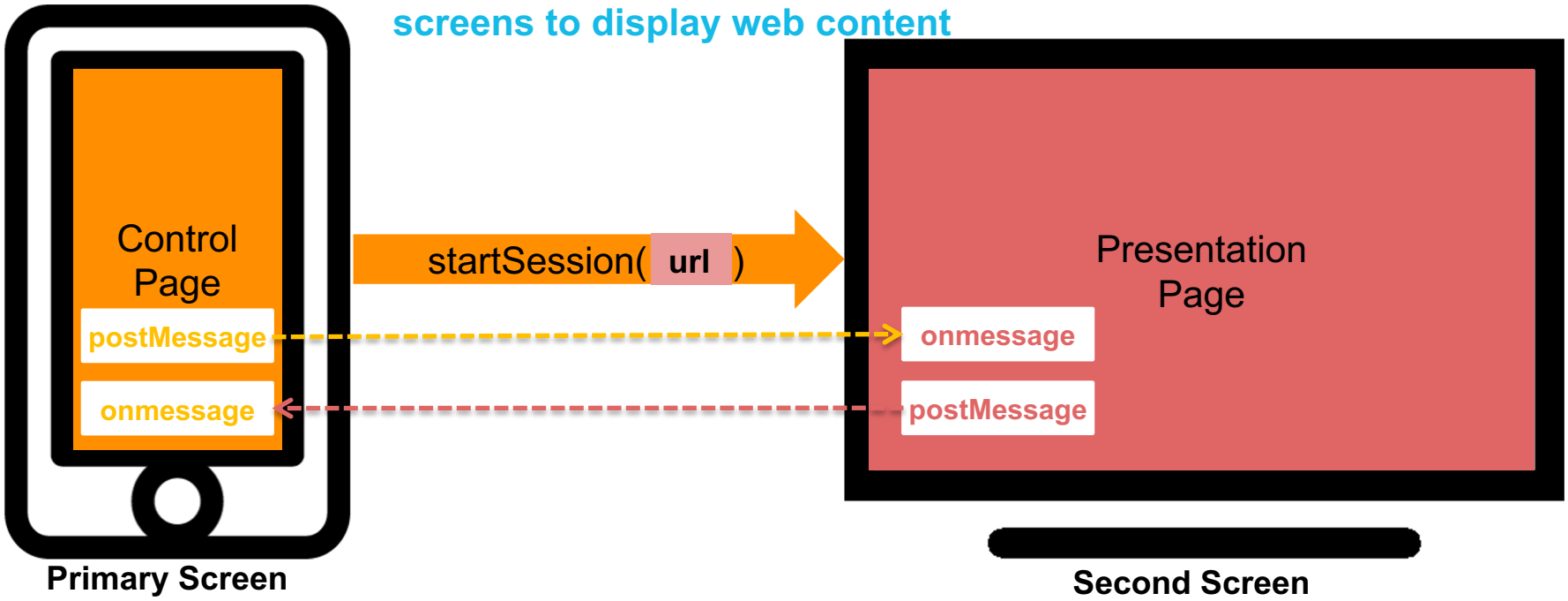
- The work of the Presentation API is continued in a Working Group
- The Working Group was created in October 2014
- The WG took the final report of the CG as initial working draft for the Presentation API
- New work items added: Remote Playback API, Window Management, Open Screen Protocol





W3C PRESENTATION API

Goal is to define an API that enables web pages to use secondary screens to display web content



W3C PRESENTATION API

Scope

- Define an API that allows a web application to:
  - ... request display of web content on a connected display
  - ... communicate with and control the web content
  - ... identify whether at least one secondary screen is available for display
- The web content may comprise HTML documents, web media types such as images, audio, video, or application-specific media
- The specification includes security and privacy considerations

PRESENTATION API EXAMPLE

```
<!-- controller.html -->
<button id="presentBtn" style="display: none;">Present</button>
<script>
  // The Present button is visible if at least one presentation display is available
  var presentBtn = document.getElementById("presentBtn");
  // It is also possible to use relative presentation URL e.g. "presentation.html"
  var presUrls = ["https://example.com/presentation.html",
                  "https://example.net/alternate.html"];
  // show or hide present button depending on display availability
  var handleAvailabilityChange = function(available) {
    presentBtn.style.display = available ? "inline" : "none";
  };
  // Promise is resolved as soon as the presentation display availability is
  // known.
  var request = new PresentationRequest(presUrls);
  request.getAvailability().then(function(availability) {
    // availability.value may be kept up-to-date by the controlling UA as long
    // as the availability object is alive. It is advised for the Web developers
    // to discard the object as soon as it's not needed.
    handleAvailabilityChange(availability.value);
    availability.onchange = function() { handleAvailabilityChange(this.value); };
  }).catch(function() {
    // Availability monitoring is not supported by the platform, so discovery of
    // presentation displays will happen only after request.start() is called.
    // Pretend the devices are available for simplicity; or, one could implement
    // a third state for the button.
    handleAvailabilityChange(true);
  });
</script>
```

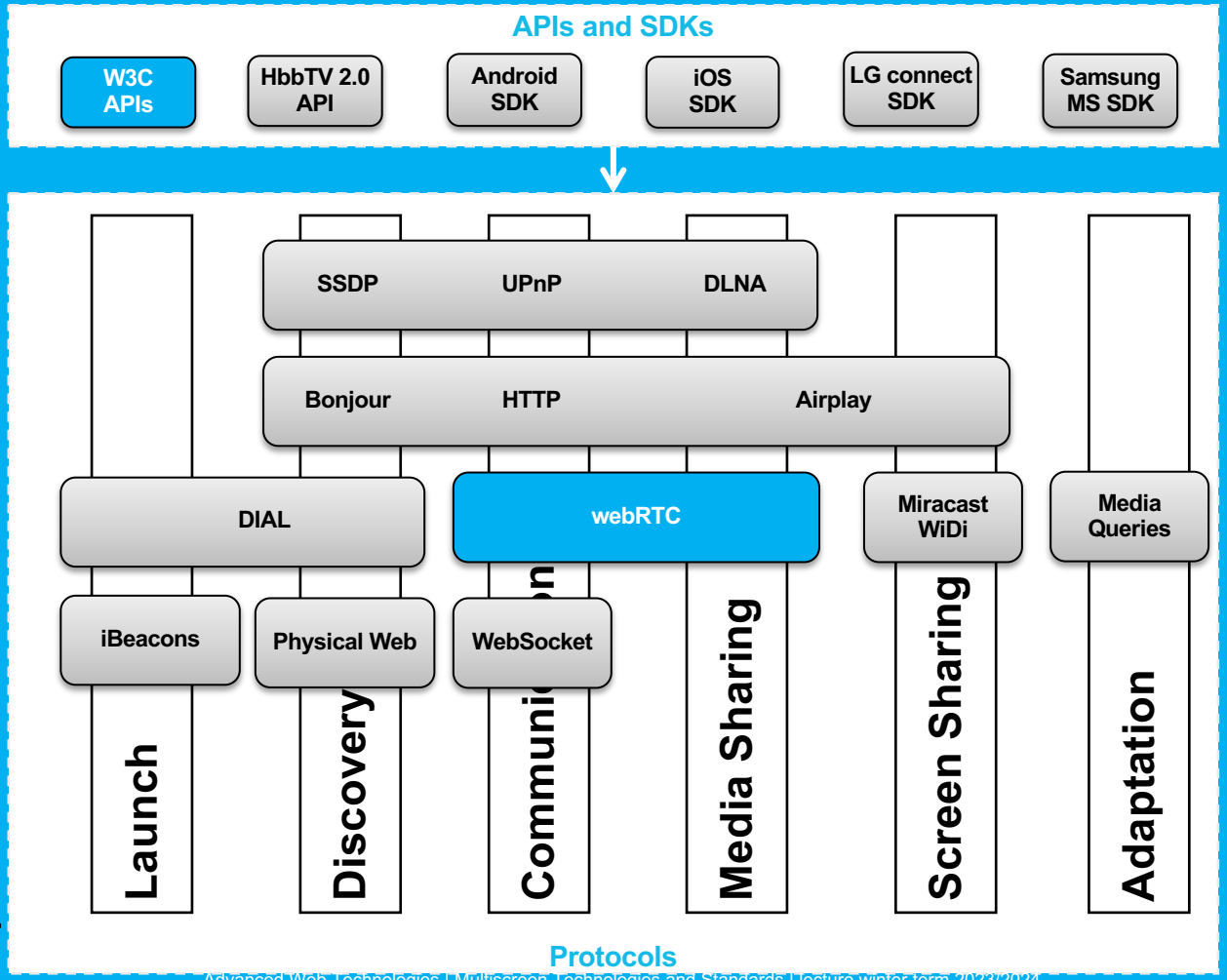
```
<!-- controller.html -->
<script>
  presentBtn.onclick = function () {
    // Start new presentation.
    request.start()
      // The connection to the presentation will be passed to setConnection on
      // success.
      .then(setConnection);
      // Otherwise, the user canceled the selection dialog or no screens were
      // found.
  };
</script>
```

```
<!-- presentation.html -->
<script>
  var addConnection = function(connection) {
    connection.onmessage = function (message) {
      if (message.data == "Say hello")
        connection.send("hello");
    };
  };

  navigator.presentation.receiver.connectionList.then(function (list) {
    list.connections.map(function (connection) {
      addConnection(connection);
    });
    list.onconnectionavailable = function (evt) {
      addConnection(evt.connection);
    };
  });
</script>
```

REMOTE PLAYBACK API EXAMPLE

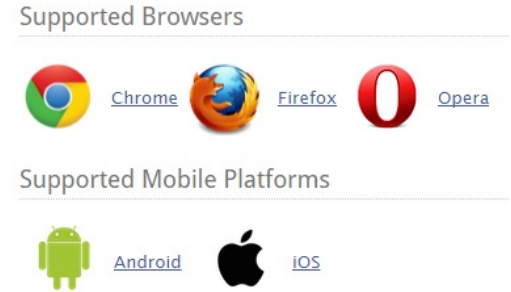
```
<!-- player.html -->
<!-- The video element with custom controls that supports remote playback. -->
<video id="videoElement" src="https://example.org/media.ext" />
<button id="deviceBtn" style="display: none;">Pick device</button>
<script>
  // The "Pick device" button is visible if at least one remote playback device
  const deviceBtn = document.getElementById("deviceBtn");
  const videoElem = document.getElementById("videoElement");

  function availabilityCallback(available) {
    // Show or hide the device picker button depending on device availability.
    deviceBtn.style.display = available ? "inline" : "none";
  }

  videoElem.remote.watchAvailability(availabilityCallback).catch(() => {
    // Availability monitoring is not supported by the platform, so discovery o
    // remote playback devices will happen only after remote.prompt() is called
    // Pretend the devices are available for simplicity; or, one could implemen
    // a third state for the button.
    deviceBtn.style.display = "inline";
  });
</script>
```

```
<!-- player.html -->
<!-- The video element with custom controls that supports remote playback
<video id="videoElement" src="https://example.org/media.ext" />
<button id="deviceBtn" style="display: none;">Pick device</button>
<script>
  // The "Pick device" button is visible if at least one remote playback
  const deviceBtn = document.getElementById("deviceBtn");
  const videoElem = document.getElementById("videoElement");

  function availabilityCallback(available) {
    // Show or hide the device picker button depending on device availabi
    deviceBtn.style.display = available ? "inline" : "none";
  }

  videoElem.remote.watchAvailability(availabilityCallback).catch(() => {
    // Availability monitoring is not supported by the platform, so disco
    // remote playback devices will happen only after remote.prompt() is
    // Pretend the devices are available for simplicity; or, one could im
    // a third state for the button.
    deviceBtn.style.display = "inline";
  });
</script>
```

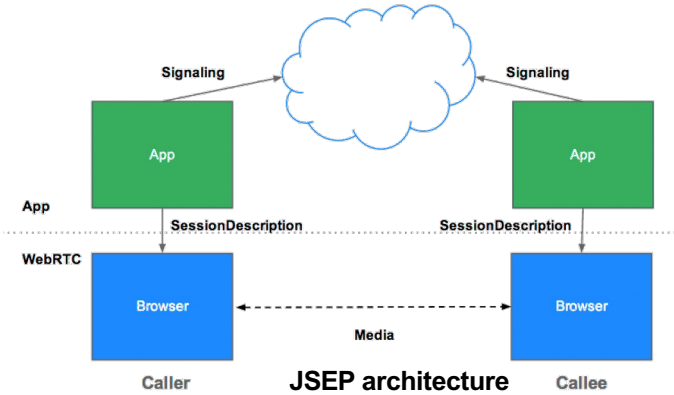

APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

WEBRTC

- W3C specification for real-time audio/video communication in the browser
  - IETF specifies **RTCWeb**, including **JSEP** (JavaScript Session Establishment Protocol) for session establishment
- Key features
  - **MediaStream** – audio and video capturing using **mic**, **camera** and **screen** → Rendering of media streams in **<audio />**, **<video />**, all CSS3 goodness
  - **PeerConnection** – Peer-to-peer audio/video connection
  - **DataChannel** – Peer-to-peer application data transfer, e.g. JSON send(), onmessage(). Same methods can be used to send/receive Blobs, etc.

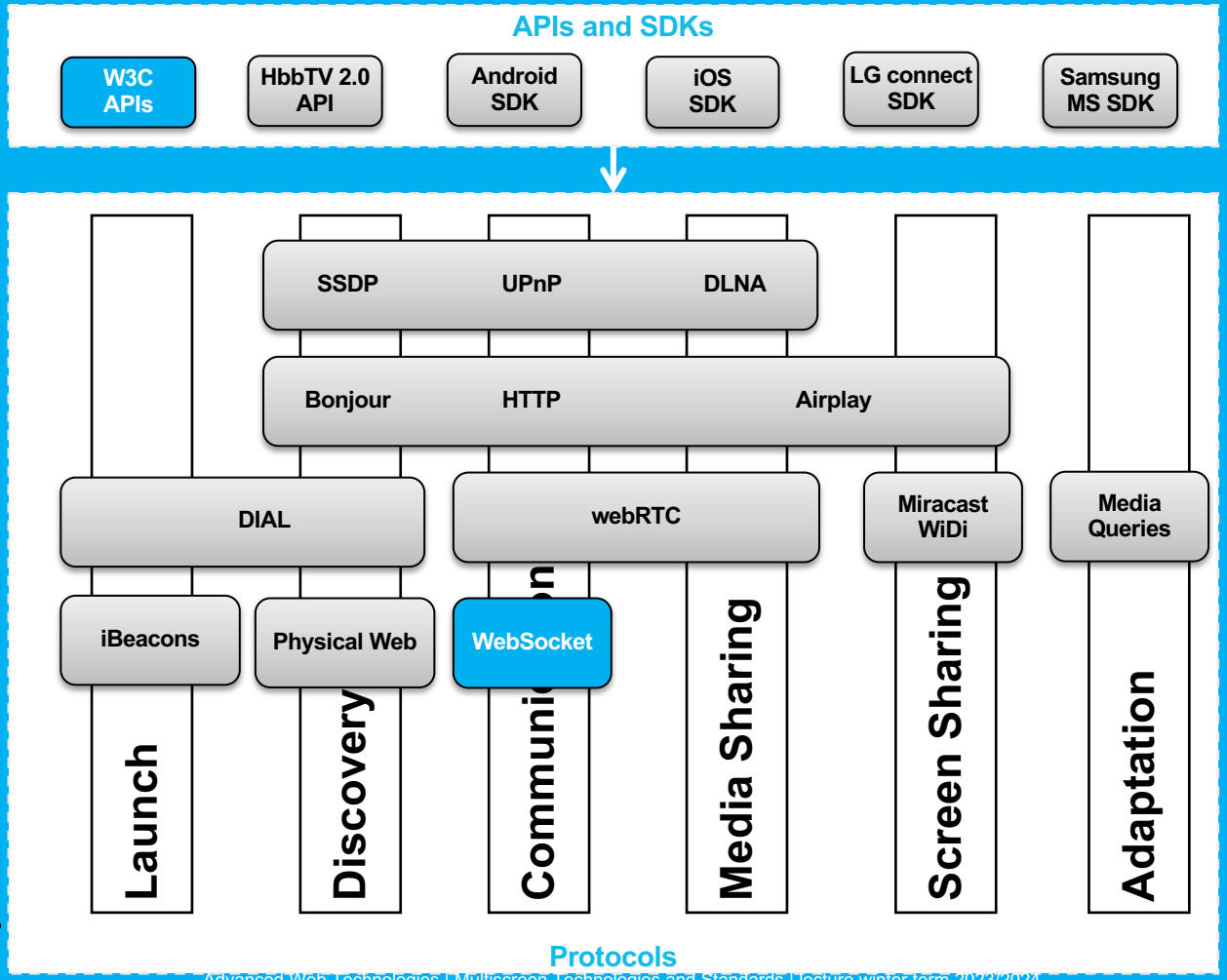


Source: http://www.webrtc.org/



JSEP architecture

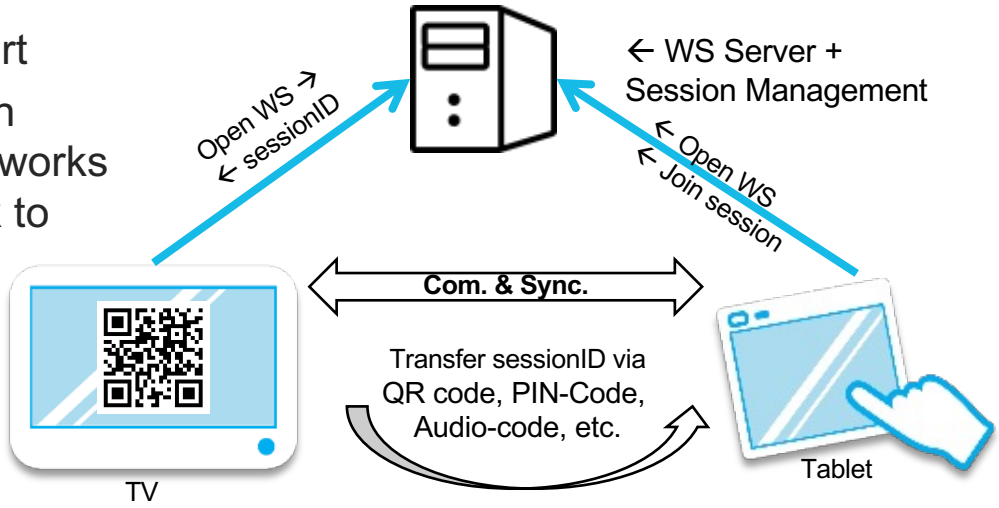
Source: http://www.html5rocks.com/en/tutorials/webrtc/infrastructure

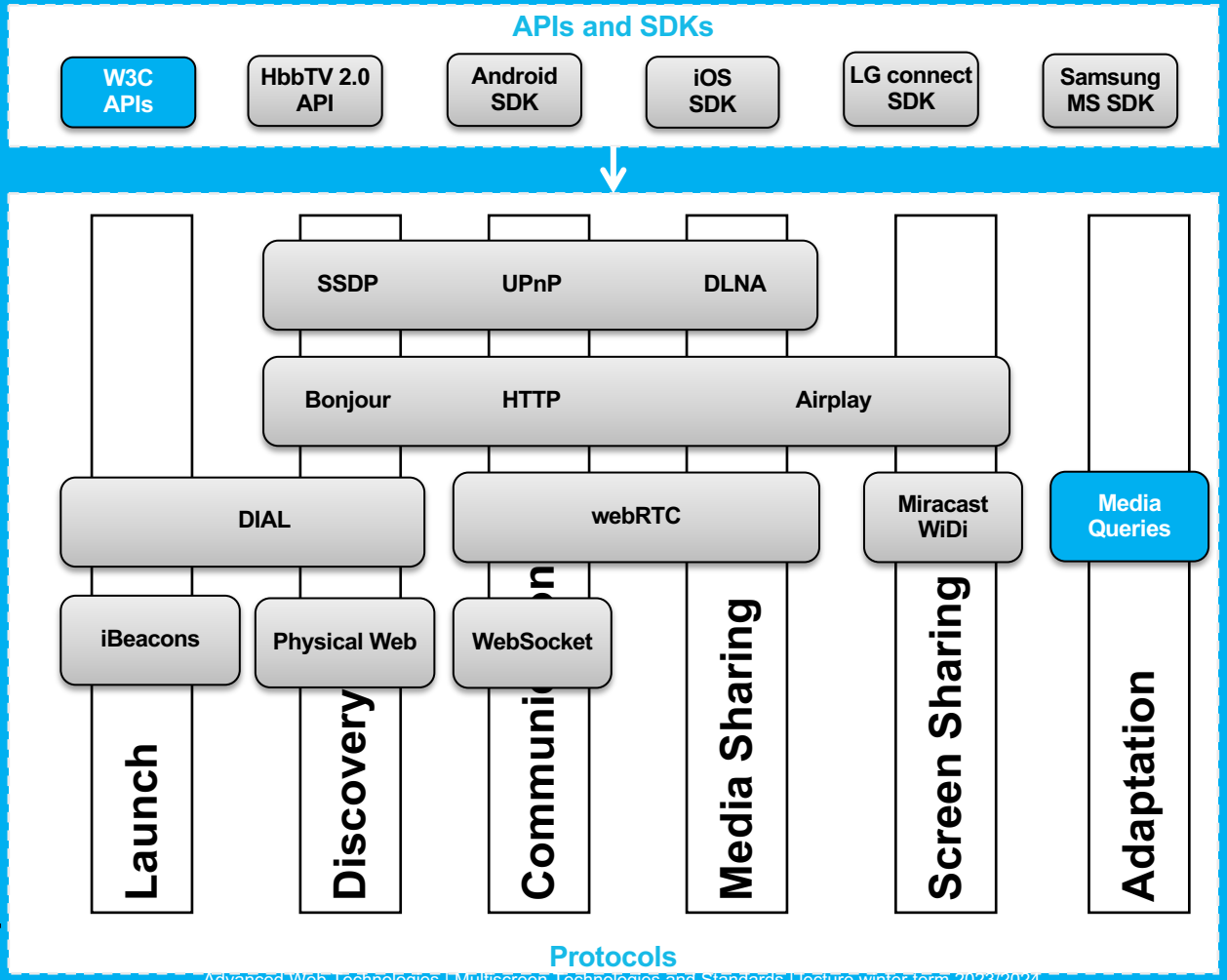

APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Media Sharing
Screen Sharing
Adaptation
Protocols

WEBSOCKET

- **WebSocket Protocol** (defined by IETF) enables **two-way communication** between a **client and server** → https://tools.ietf.org/html/rfc6455
- W3C **WebSocket API** enables Web pages to use the WebSocket protocol → http://www.w3.org/TR/websockets/
- **DOMString**, **Blob**, **ArrayBuffer** Support
- Implemented in all modern Browsers on Desktop and Mobile → 3rd party Frameworks (e.g. socket.io) can be used as fallback to support older browsers

Typical Multiscreen usage

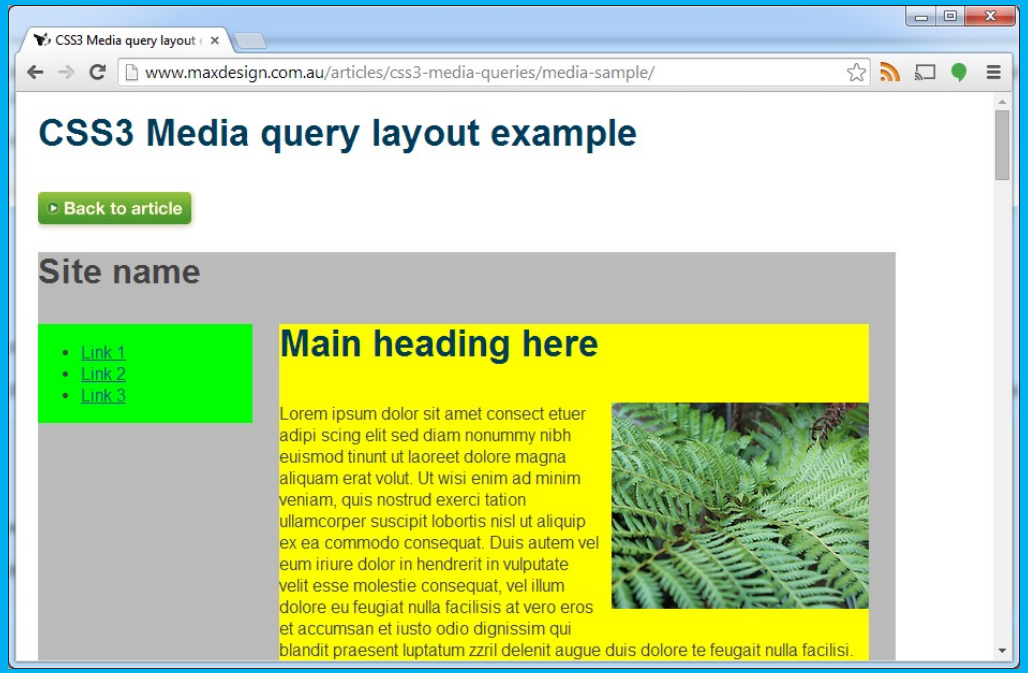



APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

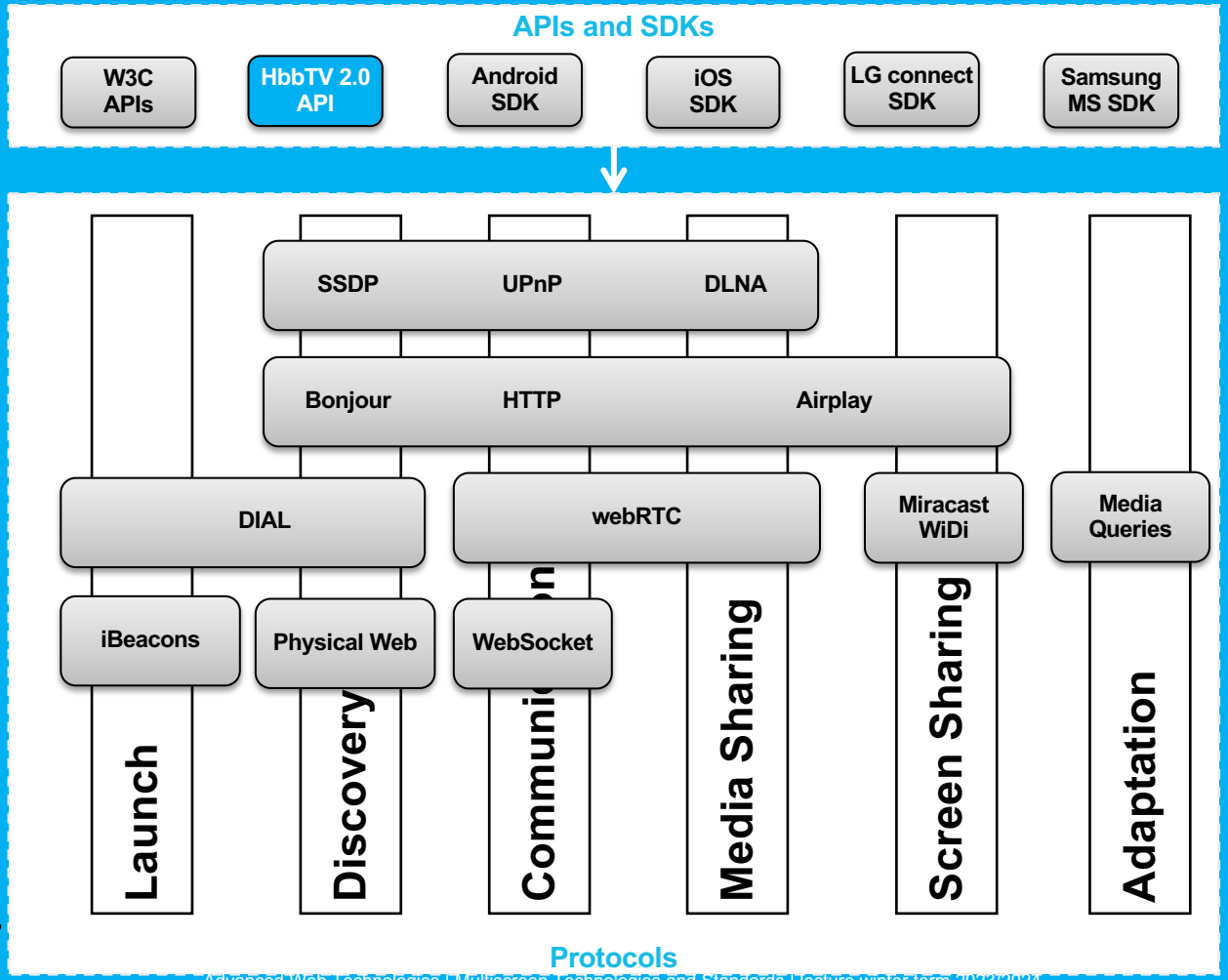
W3C CSS3 MEDIA QUERIES

- W3C Recommendation 19 June 2012 → http://www.w3.org/TR/css3-mediaqueries
- A media query consists of a media type (e.g. screen, print, speech, etc.) and zero or more expressions that check for the conditions of particular media features.
- Examples:
  - ***<link rel="stylesheet" media="screen and (color), projection and (color)" href="example.css">***
  - ***@import url(color.css) screen and (color);***
  - ***@media screen and (min-width: 400px) and (max-width: 700px) { /*CSS*/}***
  - ***@media handheld and (min-width: 20em), screen and (min-width: 20em) {/*CSS*/}***
  - ***@media screen and (device-aspect-ratio: 16/9) {/*CSS*/}***
  - ***@media print and (min-resolution: 118dpcm) {/*CSS*/}***

Simple CSS3 Media Queries Example

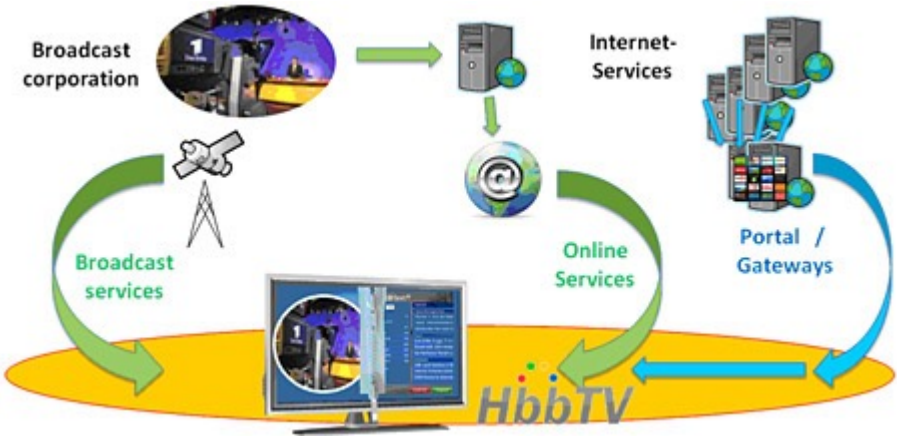


http://www.maxdesign.com.au/articles/css3-media-queries/media-sample/


APIs and SDKs
W3C APIs
HbbTV 2.0 API
Android SDK
iOS SDK
LG connect SDK
Samsung MS SDK
SSDP
UPnP
DLNA
Bonjour
HTTP
Airplay
DIAL
webRTC
Miracast WiDi
Media Queries
iBeacons
Physical Web
WebSocket
Launch
Discovery
Communi
Media Sharing
Screen Sharing
Adaptation
Protocols

HBBTV

- **Hybrid Broadcast Broadband TV** → “European initiative aimed at harmonising the broadcast and broadband delivery of entertainment to the end consumer through connected TVs and set-top boxes” (https://hbbtv.org)
- **HbbTV 2.0** is the most recent version of the HbbTV specification. It has been published by HbbTV in early February 2015.(older versions is 1.0, 1.5)



Source: http://hbbtv.org

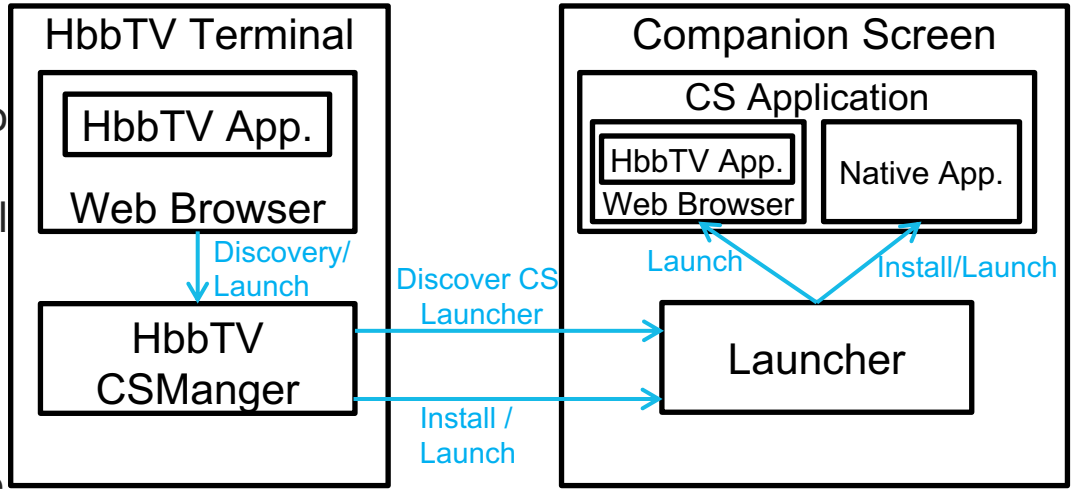
NEW FEATURES SUPPORTED IN HBBTV 2.0

- HTML5 and associated technologies: many modules from CSS3, DOM3, Web Sockets, Web Messaging, Canvas 2D, Web Workers, Web Storage, …
- HEVC video
- Subtitles for broadband delivered content
- Advert insertion into VoD content (Support of multiple Video Elements)
- **Companion Screen Features:**
  - Launching a Companion Screen Application from the HbbTV 2.0 App on the TV
  - Application to Application (App2App) Communication
  - Remotely Launching an HbbTV 2.0 Application from an App running on a companion device e.g. Tablet or Smartphone
  - Synchronizing Applications and Content across devices. This allows an app on the smartphone or tablet to synchronize with the video being presented by the TV or STB

Source: http://hbbtv.org

HBBTV2.0 CS: LAUNCHING A COMPANION SCREEN APPLICATION

- The HbbTV 2.0 App uses the HbbTVCSManager interface to discover CS devices with a running Launcher and to install or Launch native or web CS Apps
- Communication protocol between HbbTVCSManager and Launcher is not part of the HbbTV 2.0 Spec



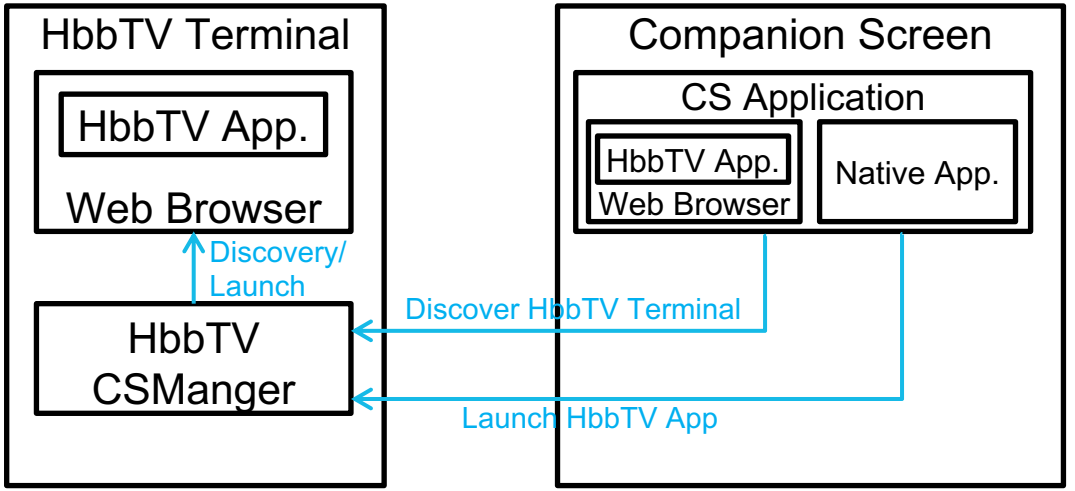
- The HbbTV 2.0 App obtains platform information, including the OS that the Companion Screen is running.

http://hbbtv.org/pages/about_hbbtv/HbbTV_specification_2_0.pdf

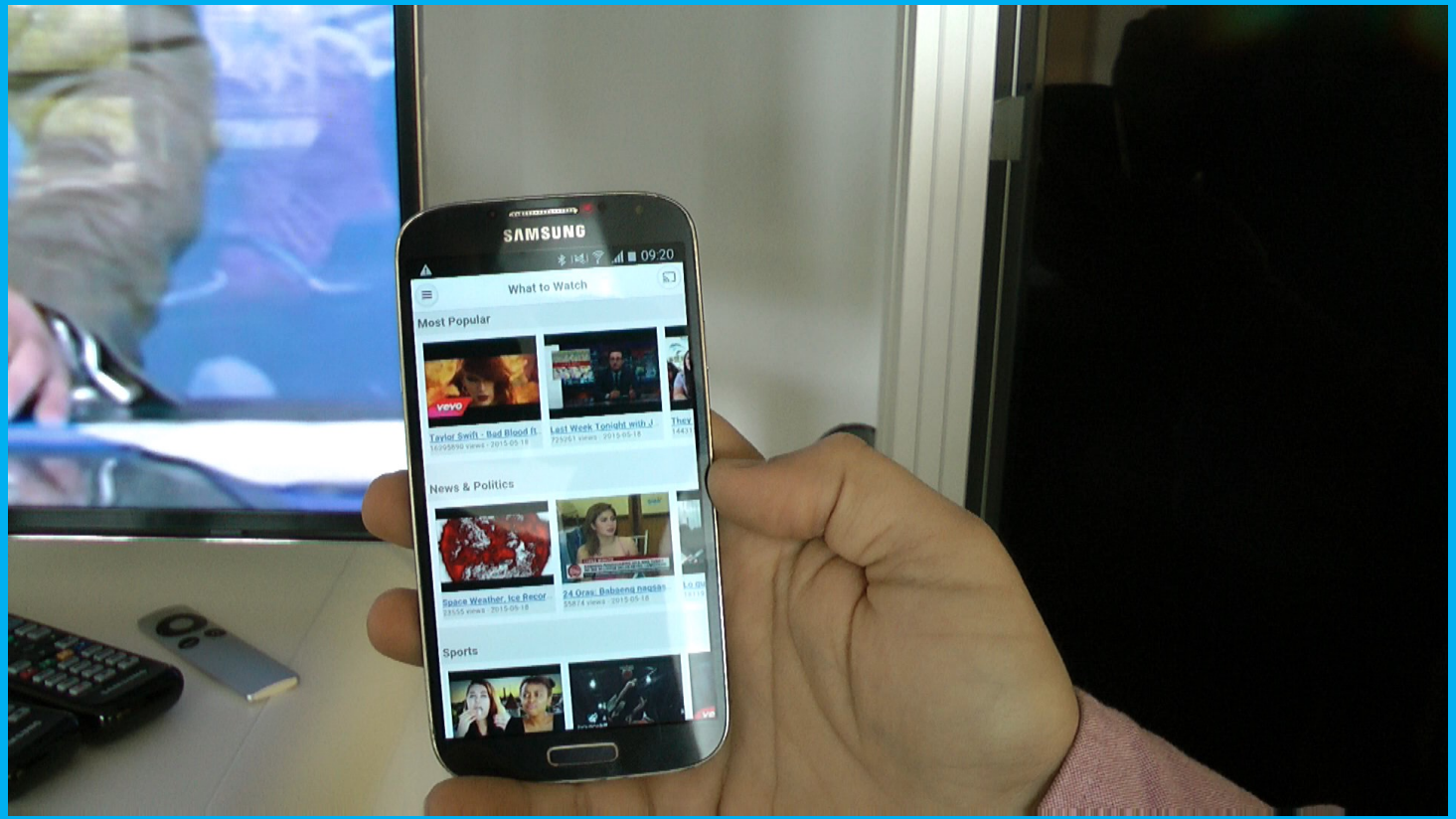

Fraunhofer
FOKU
Intro
The
Fut

HBBTV2.0 CS: LAUNCHING AN HBBTV 2.0 APPLICATION

- Launch Broadcast independent HbbTV App
- HbbTV Terminal exposes itself as DIAL App “HbbTV”
- Companion Screen App discovers (using SSDP) available DIAL servers and checks if “HbbTV” is available
- Companion Screen App launches the HbbTV App by sending HTTP POST request to the DIAL Endpoint
- The body of the HTTP POST request is an XML that contains information about the HbbTV App to launch

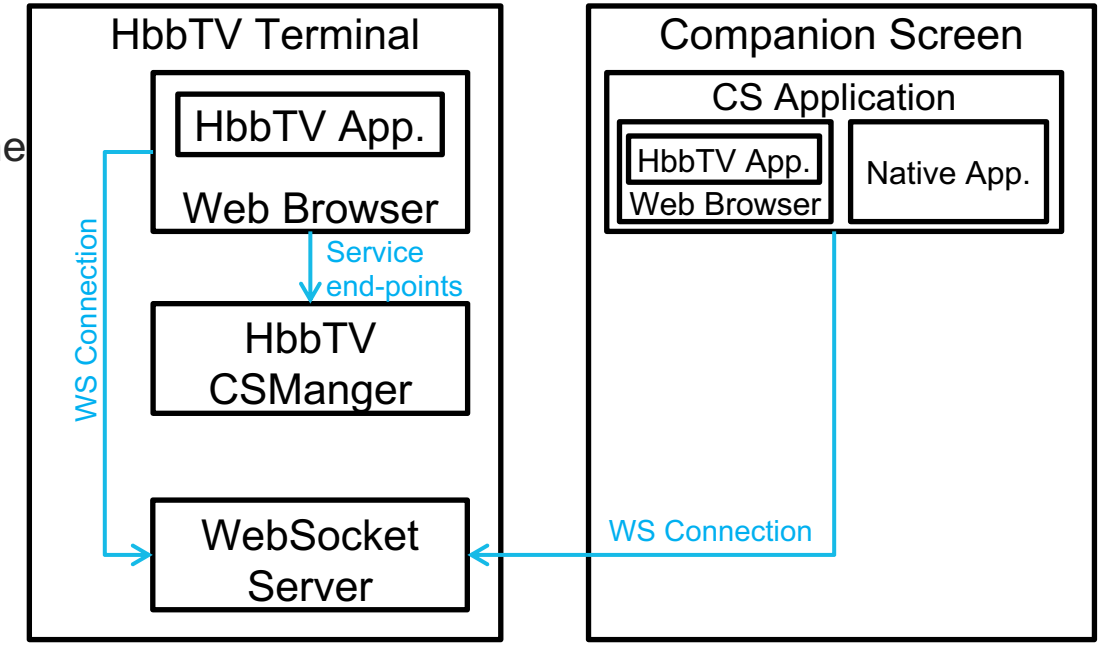


http://hbbtv.org/pages/about_hbbtv/HbbTV_specification_2_0.pdf


SAMSUNG
09:20
What to Watch
Most Popular
Taylor Swift - Bad Blood ft.
Last Week Tonight with J
News & Politics
Space Weather, Ice Recor
24 Oras: Babaeng naggas
Sports

HBBTV2.0 CS: APPLICATION TO APPLICATION COMMUNICATION

- The HbbTV Terminal runs a Websocket Server
- App2App end-point contains the address of the WS Server
- The HbbTV App requests the App2App end-point from the HbbTVCSManager
- The CS App gets the App2App end-point after DIAL App Launch
- The HbbTV and CS Apps establish WS connections

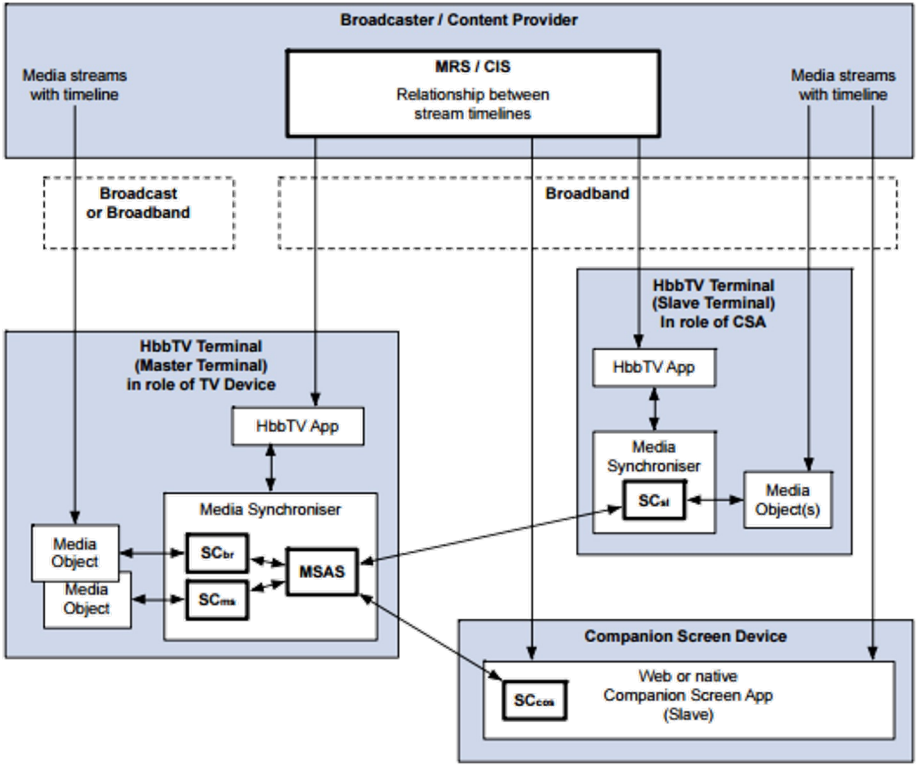


http://hbbtv.org/pages/about_hbbtv/HbbTV_specification_2_0.pdf

- The Websocket Server acts as relay and pairs the WS connections on the same channel

HBBTV2.0 CS: SYNCHRONIZING APPLICATIONS AND CONTENT

- **Multi-stream synchronization**: defines how streams delivered via broadcast and broadband can be synchronized
- **Synchronizing applications and content across devices**: This allows an app on the smartphone or tablet to synchronize with the video being presented by the TV or STB based on the DVB Companion Screen (“CSS”) specification.
- Application and content synchronization builds on the “TEMI” timeline recently standardized in MPEG



Q&A

THANK YOU FOR YOUR ATTENTION