Remote screens
with user-friendly pairing and optional ad support

Feb. 2014

Motivation/Target

Smartphones and tablets became powerful computing devices and their capabilities are increasing further. There already is a vast ecosystem of applications which is getting larger from day to day. Devices like smartphones and tables store applications, store contacts, and store other data. They become a central part and the central hub of “digital life”.
On the other hand, smartphones and tablets have a restricted screen size (or other limitations with input/output devices as well). In a typical household there are devices with larger screens, e.g. PC monitors or televisions or more powerful input/output devices. It is a logical step to not only regard all these as several, separate devices that perhaps can share data with each other (e.g. via central media server or via cloud storage). If smartphones and tablets really become the center of “digital life”, it makes sense to run the applications on these devices and use the bigger screens of other devices like televisions for output.
In the long run, it seems to be the far better approach instead of having independent devices. It avoids the need to sync applications, appearance, and data. It is much more user-friendly since the user always has “his” environment available.
Future might even make desktop PCs as we have today obsolete. Smartphones/tablets will have more than enough capabilities to run most applications that need the resources of a desktop PC today. Therewith, PCs are no longer needed as computing devices running applications. Only the peripherals like big monitors, input devices (e.g. desktop keyboard, scanners) and output devices (e.g. printers) are still needed for doing work that is done on desktop PCs today.
Already today, we see the shortcomings of having separate devices having applications on their own, e.g. with SmartTVs: The user needs to learn to use different environments, there are proprietary app stores, the devices have security vulnerabilities requiring the user to pay attention, and it is difficult to share data between several devices. The SmartTV vendors created apps to remote control their environments from smartphones/tablets – but that’s it already. Future needs to and will be different: The smartphone is NOT to be used as an input device (remote control) for a smart TV. The smart TV has to be just an output device for the smartphone/tablets instead.
Note that the motivation above focusses on smartphones/tablets but is not limited to such devices (other computing devices like watches, glasses, or whatever are conceivable).

Challenge (technical requirements)

The screen of the smartphone/tablet shall be mirrored to another screen, e.g. a computer monitor or a TV screen. In an advanced approach, for some applications it makes sense to use the screen as a separate second screen instead of using it as a mirror of the primary one. Similarly, other input/output devices may be virtually attached.

The solution shall work network-independently. Even if smartphone/tablet and the devices with the screen are not in the same network, e.g. in a hotel, sharing the screen and input devices shall be possible. The network shall, at least optionally, be regarded untrusted so that authentication of the communicating peers and encryption between the communicating peers shall be supported.
Screen devices from multiple vendors shall be supported; vendors shall not be required to implement support for the solution to work.
Pairing devices, i.e. the smartphone/tablet and the device with the screen, shall be user-friendly and self-explaining. A temporary pairing shall be possible so that also screens that are not owned by the smartphone/tablet user can be used (e.g. the TV in a hotel). The pairing process shall take security into account, i.e. it shall be ensured that such a pairing takes place between the intended devices only and in a controlled manner.
Besides just using the screens of other devices, also the input devices (e.g. keyboard, mouse, webcam) of other devices shall be usable. This makes the solution versatile and generic. The user can make use of all available devices and select the ones suited best for the current task to be done. Similarly, other output devices besides monitors (like printers) can be connected.
To create revenue, the possibility to show ads to the user shall (at least optionally) be exploited. Possibilities to show ads targeted to the user's preferences shall be exploited, too.

Related Work

There is no user-friendly, comprehensive solution to the presented problem known to the inventor. However, several technical building blocks are known well. Further, there is software using some of the building blocks to solve other, partly related problems.

Remote desktop protocols

Remote desktop protocols are used to share the screen or a part of the screen from one device to another via a network. Additionally, input devices (like keyboard and mouse), other output devices (like printers or audio devices), as well as special other entities (like network shares or clipboard) can be forwarded/shared using separate channels in many of the available protocols.

Already in the 1980th, the "X Window System" (X11) was developed. It is a protocol and software providing "windows" on graphic user interfaces. It uses a client-server model and provides network transparent communication between client and server. This enables windows of applications running on one device (usually a central server in former times) to be shown on another device (usually a dumb terminal in former times). Due to its design, the protocol is not suited to be used over networks with low bandwidth and high latency (like wide area networks / the Internet).

In the 1990th, "VNC" (Virtual Network Computing) was developed. It uses the RFB (remote frame buffer) protocol to forward the screen of one device to another client device. Some input devices (keyboard and mouse) of the client device can optionally be used to control the server-side device. Due to the openness of the protocol, it is widely used as the basis for other software and developments. However, the performance over slow network links is not very good and fast screen updates (e.g. for videos) are very limited.

Citrix developed with “ICA” (Independent Computing Architecture) a proprietary protocol for similar purposes. It was more efficient than VNC, especially over low-bandwidth networks. The technology was licensed by Microsoft. That company implemented “RDP” (remote desktop protocol) to enable remote access to a Windows desktop. The protocol is used as the basis for Microsoft’s terminal servers.

NoMachine developed with “NX” an even more efficient remote desktop protocol. The NX software can work as a proxy and be used to expose devices running X11, VNC and RDP over low-bandwidth links.

Qumranet/Redhat developed “SPICE” (Simple Protocol for Independent Computing Environments). It is an architecture-independent protocol to make desktops accessible efficiently over networks. It even enables moving content like videos to be shown smoothly on the remote device. The NX libraries and the SPICE libraries are available as Open Source software.

Use of remote desktop protocols

There are hundreds of applications that implement and use the protocols presented above. Some of them are listed here: http://en.wikipedia.org/wiki/Comparison_of_remote_desktop_software

Most of the software aims at remote support or at implementing a screen sharing tool for conferencing solutions. Some widely known applications are TeamViewer, OpenMeetings, and WebEx.

Another part of software aims at helping in daily use of computers. Some examples: x2x (allows the console (keyboard and mouse) on one X terminal to be used to control another X terminal), ZoneScreen (“extending your desktop workspace using displays of network connected computers or portable devices like Pocket PC”; http://www.zoneos.com/zonescreen.htm), MaxiVista (use laptop as second screen for your PC; http://www.maxivista.com/de/).

For smartphones there are lots of clients for the described remote desktop protocols (e.g. VNC, RDP) available. However, server implementations are quite rare, usually limited to VNC, and seem to provide only slow screen refresh rates. Here are the links of two servers: “droid VNC server” (https://play.google.com/store/apps/details?id=org.onaips.vnc&hl=de) and “VNC server” (https://play.google.com/store/apps/details?id=com.schumi.vncs&hl=de).

Screencast protocols

AirPlay from Apple is a streaming protocol for the wireless transmission of content from Apple devices to AirPlay-capable (license needs to be obtained from Apple) receiving devices like loudspeakers, AV receivers, and TVs. This includes the possibility to show the screen content of an Apple device on another device. Apple Bonjour is used for device discovery in the local network. A similar vendor-proprietary initiative is Intel’s Wireless Display (WiDi) that requires an Intel chipset and devices that support that protocol.

Miracast is a peer-to-peer wireless screencast standard defined by the Wi-Fi Alliance. It supports video and audio. In contrast to AirPlay and Intel’s Wireless Display, it is an open and vendor-independent standard. It enables to show the screen of on device on another device. Both devices need to support Miracast and be connected to the same wireless network.

DIAL (in the context of Chromecast) stands for “DIscovery And Launch” protocol. It was co-developed by Netflix and YouTube. It is used to search for available devices on a WiFi network and exchanges information on how to connect to the device. The protocol is used by Google for the Chromecast device which is an HDMI dongle that shows content on the connected device. The primary method of playing media on the device is through Chromecast-enabled mobile apps and web apps so that it cannot be used in a generic manner. In addition, the “tab casting” feature mirrors the content of a Google Chrome browser tab (note that content that uses plug-ins like Silverlight and QuickTime is not fully supported).

Screencast protocols gain popularity with increasing support. For instance, the alternative Android firmware CyanogenMod now provides an API to simplify their implementation (see http://review.cyanogenmod.org/#/c/50449/).

Using QR codes

The QR code community is looking for and advocating ideas for the use of QR codes for various purposes. Among some other ideas, the idea of pairing devices using a QR code is found here:
http://www.qrcode.es/en/2013/04/paring-devices-qrcode/
The application discussed in that article is a presentation application that can be remote controlled using a smartphone after pairing.

This application (http://www.demobo.com/product.html) aims at enabling smartphones to remote control browser tabs and some applications running in browsers (Google Drive, slideshare, scribd, dropbox, Youtube) on other machines. Note that this differs from what shall be achieved with our solution (exposing a running application or the smartphone screen, respectively, to an external screen).

The Solution

Applications remain on the smartphone/tablet only. The TV or another output device acts as second screen. Enhance and combine existing building blocks like remote desktop protocols and messaging services. Don’t limit the solution to a local network. Implement user-friendly pairing. Consider security. Allow the solution to be combined with showing targeted ads.

Process / solution description from user point-of-view

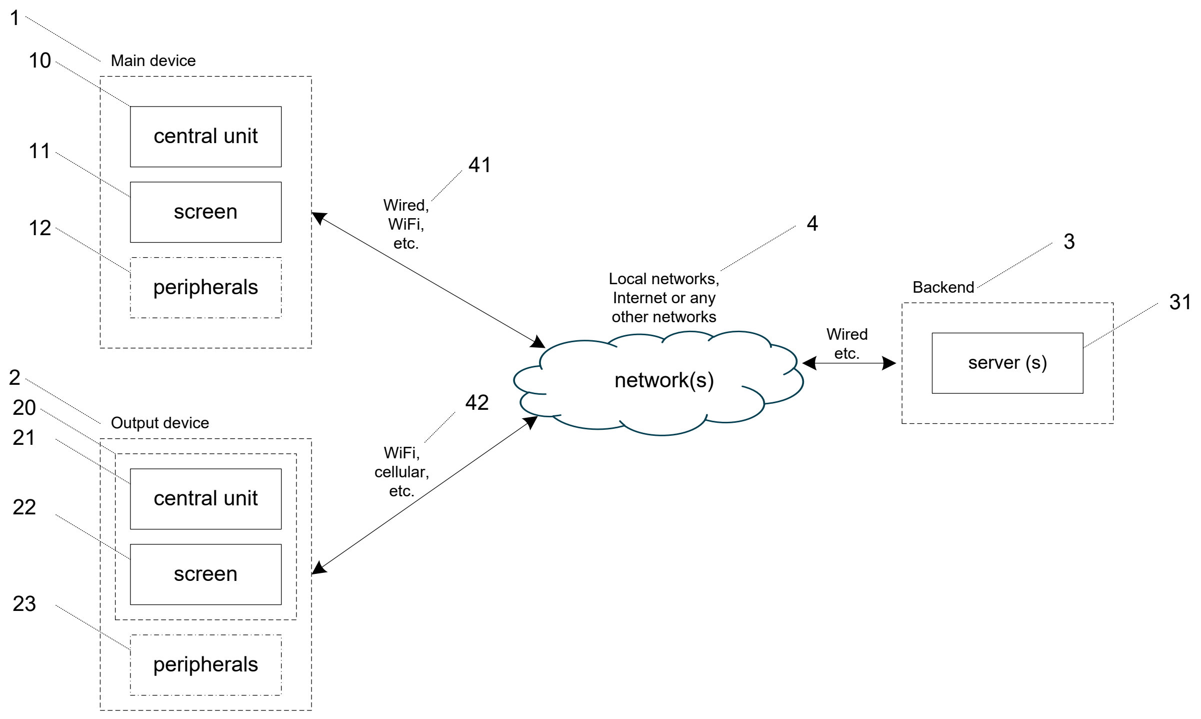
1. *Running application on “output device”:*
   A web site is loaded (or a stand-alone client started) on the output device. It shows a QR code. Optionally, also some ads can be shown. Such ads generate revenue in exchange for providing service.
2. *Running application on “main device”:*
   On the smartphone/tablet (or any other computing device), a controlling application is started. The application offers to scan a QR code. In addition as an optional feature, a contact list of already known output devices to use can be shown. Like in other smartphone apps, ads can be shown optionally, e.g. at the bottom area.

3. *Scanning the QR code:*
   When the user scans the QR code on the output device, this proves that smartphone/tablet and output device are currently in proximity.
4. *Connecting:*
   In the smartphone/tablet application, the user can now choose whether to just connect to the output device or also to open a web browser in which more information on the currently shown ad on the screen of the output device is displayed. If no ads are used, this user interaction can be skipped and the connection be established instantly.
5. *Normal work:*
   The two devices are temporarily paired with each other now. The screen of the output device shows the screen content of the smartphone/tablet. The user can now use his smartphone/tablet as usual. The screen content is mirrored to the screen of the connected output device. Special applications that are dual-head capable could also use the screen of the output device as an independent second screen.
   Optionally, peripherals (input devices like keyboard and mouse) of the connected output device can be used to control the smartphone/tablet. As an additional feature, other peripherals (like connected printers) can be virtually attached to the smartphone/tablet.
6. *Disconnecting:*
   The connection between smartphone/tablet is terminated on user request. The output device again shows a (new) QR code to allow new pairing, see the beginning of the description.

Note that the pairing using a QR code is just an example. Other means of user-friendly data transfer can be employed as well (e.g. Near Field Communication, other 2D/3D codes, or Bluetooth).

Physical setup

The solution physically consists of the following entities: A “main device” (1), an “output device” (2), and a “backend” (3). Connectivity between these three is established by a network (4).



The “main device” (1) is usually a smartphone or tablet. But it can also be any other computing device. It is the device that runs the user applications. It consists of a central unit (10) running the applications and providing connectivity, a screen or touch screen (11), and optionally additional peripherals (12; e.g. other input devices like an external keyboard).

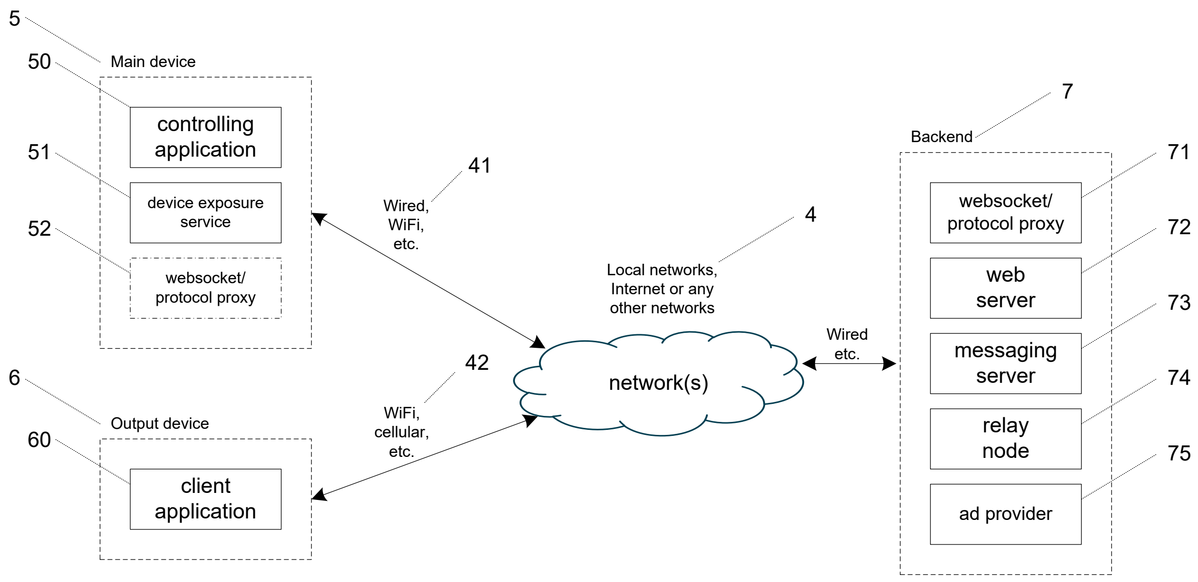
The “output device” (2) is the device whose screen shall be used to display the content of the screen of the “main device”. The “output device” consists of a central unit (20) controlling the device and providing network connectivity, a display/screen (21), and optionally peripherals (22; e.g. other input devices like mouse or keyboard). The output device can be a SmartTV (with a web browser or a stand-alone application running on it), a dongle-device (e.g. Android-based) connected to a “normal” TV or SmartTV (even an outdated one, i.e. old platform, old web browser), a dongle-device (with a web browser or a stand-alone application running on it; e.g. Android-based) connected to a PC monitor, or a normal PC/notebook (with a web browser or a stand-alone client application running on it).

The “backend” (3) consists of one or several servers running backend services. These services can be operated by a single operator or by multiple operators.

The “network(s)” (4) provides connectivity between the three entities. It is composed of LANs (local area networks) in which the devices and the backend reside and other networks like the Internet. The LANs are interconnected, in practice by the Internet. It is expected that one or more middleboxes are present in the network that may limit direct reachability to the “main device” and the “output device” from the outside, e.g. there could be home routers performing address masquerading (port address translation between inside network and the Internet). Other middleboxes like firewalls may be present but considered to be configured properly to allow required connectivity. By just relying on the http/https protocol, firewall trouble can be limited. Connectivity between the main device, the output device and the backend can be realized by arbitrary physical means in the access network, e.g. wired or by different kinds of wireless networks (41), (42).

Application setup

The physical devices (1), (2), and (3) are also shown in a second figure as (5), (6), and (7), respectively. In that figure, the software running on the physical devices/machines is shown.



The "main device" (5) runs a "controlling application" (50), a "device exposure service" (51), and optionally a "websocket / protocol proxy" (52). The "controlling application" (50) is the end user application. It is used by the user to control the solution and to interact with the solution. The "device exposure service" (51) exposes the screen of the devices using an appropriate protocol (usually a remote desktop or screencast protocol). A server implementation of SPICE is suited since it is an open protocol that is capable to process audio and video well. Another option would be to expose a WebRTC (Web Real-Time Communication; an API/standard by the World Wide Web Consortium for browser-based real-time communication) stream. (50) and (51) can also be implemented as a single application. The optional "websocket / protocol proxy" (52) is only needed if the "output device" (6) requires a different protocol so that a protocol translation is needed. Such a translation can also be done within the backend (71) but if a direct connection between the "main device" and the "output device" shall be established instead of employing an indirect connection via the backend (which would result in higher latency), the translation needs to be done on the main device.

The "output device" (6) runs a client application (60). That client application controls the interaction with the backend and the main device, renders the screen content, and gets and processes events from relevant peripheral devices. The client application can be a web browser with a client-side script running in it or a stand-alone client application that does not make use of a web browser. A stand-alone application would control the screen and the input devices and would implement a remote desktop or screencast client. The alternative would be to provide this functionality by a web browser running a client-side script in it. The client-side script could use web technologies (WebRTC) or implement a different protocol. The stand-alone client has performance benefits, but an implementation is required for each platform increasing efforts. The browser-based client application can be used on any platform that provides an up-to-date web browser (so that current technologies like websockets and WebRTC are available).

The “backend” (7) consists of a websocket/protocol proxy (71), a web server (72), a presence server (73), a relay node (74), and an ad server (75). Some of these are optional, see the further description.

A “websocket/protocol proxy” (71) is needed when protocol conversions are needed (e.g. when a browser-based client application using websockets is used on the output device but a different approach on the main device. Then the proxy converts protocol messages into html protocol messages that can be transmitted to and from web browsers). For instance, if the “main device” uses a remote desktop protocol like SPICE, the proxy can act as a SPICE client and provide events via http websockets to the client application running in an html5-capable web browser.

The “web server” (72) provides content to the “main device” and the “client device” whenever needed. Any other means for content delivery could be used as well. However, using a web server ensures that the devices do not have trouble connecting since the http/https protocol is usually allowed in firewalls.

The “messaging server” (73) is an instant messaging server where clients can register their presence and then exchange messages with each other. For instance, an XMPP (Extensible Messaging and Presence Protocol) server like OpenFire can be used. Any other messaging platform, e.g. one based on the IMS (IP multimedia subsystem), can be used as well.

The “relay node” (74) provides connectivity between the “main device” and the “output device” for the case that a direct connection between these devices is not possible, e.g. due to firewalls or other middleboxes. If an XMPP server is used as a “presence server”, the relay node can be a so-called “Jingle node”. Jingle is an XMPP standard that includes relay node service.

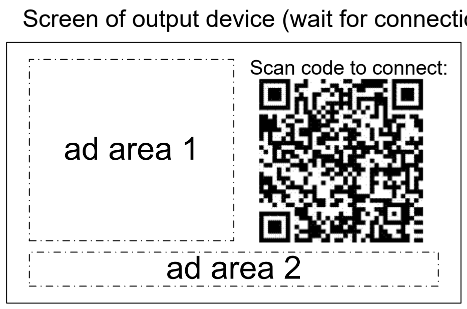
The “ad provider” (75) provides content for advertisements. It is optional and only needed in case ads shall be shown to the user in order to create revenue for the use of the solution. Note that the ads can be target-specific (using information about the user) and context-specific (using information about the environment/context of the user, e.g. devices, location, time of day).

Process / solution description from a technical point-of-view

In the following exemplary description, it is assumed that a browser-based client capable of using WebRTC on the output device is used. However, the other variants described above are possible as well. It is also assumed that an XMPP messaging server is used. It is also assumed that QR codes are used for encoding data into a graphical representation. Of course, the other options mentioned in this document are possible as well. For instance, any other appropriate coding scheme can be used instead of QR codes, too.

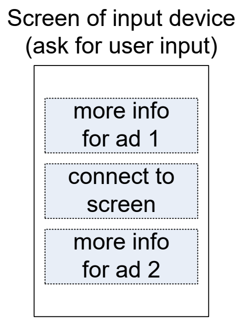
1. *Running application on “output device”:*
   The web browser on the output device loads a web page from the web server. Any functionality needed is provided by a server-side application (e.g. php or Python web

application) and the client-side implementation running in the web browser (e.g. html5/Javascript).
The following tasks take place:
The output device is represented as a registered XMPP user in the XMPP messaging server. The presence status of that user is set to “online”. Communication with the XMPP user that represents the output device is now possible via the messaging server.
Instead of registering a new (temporary) XMPP user on each invocation, a browser cookie with the registered XMPP identity can be stored on the output device. In the next invocation, the XMPP user registered previously can be reused. Reusing the XMPP user also helps in creating a permanent output device identity which can help in target-specific and context-specific advertising.
The XMPP user and an additional security token is coded in an URI and shown on the web page encoded as a QR code. The security token and therewith the QR code needs to change regularly (e.g. every 60 seconds). That temporarily valid security token is used to ensure that old URIs cannot be used any more (to avoid connecting to the output device without being / having been physically present within the validity time window). For instance, such an URI could look like this: “xmpp:dev12345678@xmpp.telefonica.com?time=2013-08-07T16:32&token=a840fef347a29e”. The security token can in the simplest form be a random number but more complex schemes can be used.
As an extension, the use of an URL instead of an URI is recommended. It could look like this: “https://xyz.telefonica.com/?xmpp=dev12345678?time=2013-08-07T16:32&token=a840fef347a29e”. The data from the URI is contained within the query string and can be extracted from there. When surfing to the web page, a manual describing how to download the smartphone/tablet application and more information on how to use the solution can be shown. This way, helpful information is shown to the user independently of the specific QR code scanning app used by him.
The encoded URI/URL is shown on the screen of the output device.



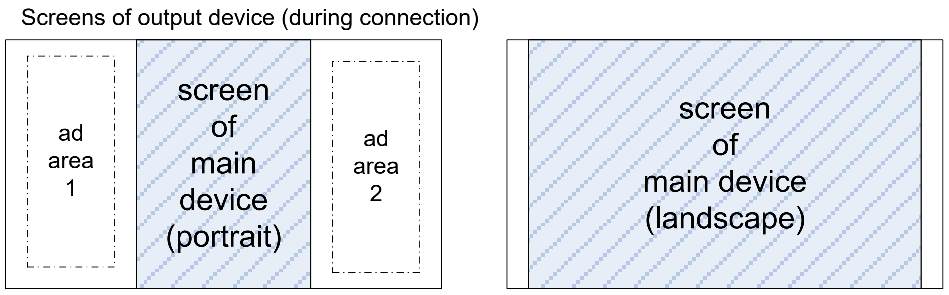
Optionally, ads can be shown in the spare area of the screen of the output device. If a permanent XMPP user is used and information on the devices/user that connect to the output device is logged and analyzed in the backend, ads can even be targeted to the user’s personal preferences.
Instead of a QR code, an NFC tag attached to the output device could be scanned to retrieve the URL. Since this URL can only be static for cheap NFC tags, an interactive handshake procedure is needed (e.g. some text is shown on the output device that needs to be entered on the main device to prove proximity).

2. *Running application on "main device":*
   The application (controlling application) is expected to be a stand-alone application since it needs access to the device's hardware. After being started, the application connects to the XMPP server in the backend and sets its presence status to "online". On first usage, a new XMPP user needs to be created beforehand. The application has the capability to scan QR codes.
   Ads can be shown within the application. They are retrieved from the ad server in the backend (via http/https).
3. *Scanning the QR code:*
   When the user scans the QR code shown on the screen of the output device, the QR code data is decoded by the application. Now the application knows the XMPP user of the output device and the current security token.
4. *Connecting:*
   Via XMPP messaging between the controlling application on the main device and the application on the output device (or the representing server-side script, respectively), the connection details (e.g. supported protocols, screen sizes, network reachability) are negotiated and the validity of the security token is checked. With the negotiated parameters, the screencasting and optionally the forwarding of input devices is established. In the assumed setup, the device exposure service acts as a remote desktop server / screencasting server, the application on the output device acts as client. An appropriate protocol that also supports video is used, e.g. SPICE. The screen updates, optionally audio, and device events can be encapsulated in the html protocol (using WebRTC) to support browser-based clients and to avoid firewall restrictions. An encapsulation / protocol conversion can be done by the websocket/protocol proxy when main device and output device use different protocols. A direct connection that is not redirected via the backend is preferred to avoid backend load and latency. If a connection needs to be redirected via the backend since direct connectivity is not available, relay nodes are used for this purpose.



   In case ads are shown besides of the QR code, it makes sense to give the user a choice whether to just connect or whether to connect and open a web browser with more information regarding the currently shown ad (in case of multiple ones, the user can choose). For this, in the negotiation also information regarding the currently shown ads on the output device is sent to the controlling application on the main device. With that data, a selection box can be shown to the user with which he either can select to just connect or to connect and retrieve product information.

5. *Normal work:*
   The controlling application goes into the background. As already described in the user-perspective, the user can now use his device as usual, seeing the screen output also on the screen of the output device. The remote desktop / screencast protocol takes care of the screen updates and also of the forwarding of the output device's input devices or even output devices (if present) to the main device.


Screens of output device (during connection)
ad area 1
screen of main device (portrait)
ad area 2
screen of main device (landscape)

6. *Disconnecting:*
   The user brings the application into the foreground again and selects to disconnect. The running connections are shut down and the applications on both devices go back into the initial state (but keeping their presence status with the XMPP server).

All the network traffic should be encrypted (e.g. using SSL) and the peers should authenticate themselves (e.g. using certificates). Ideally, the need for protocol proxies in the backend should be avoided so that data can be encrypted end-to-end. Implementation should be in a way that there is no need to trust the backend or the network.

In the example above, the connection is initiated from the main device. If the output device and the main device trust each other (e.g. both have the same owner; in any case: trust relationship needs to be established beforehand), connections initiated from the output device can be allowed as well thus improving usability.

Extensions / Other Use Cases

Extension to support main device without input devices

In the presented solution, the user's main input device is on the main device. Peripheral devices of the output device can be used optionally just as if they were directly connected to the main device (using a remote desktop protocol).

However, also main devices without any input devices (or with insufficient input devices) can be used. Instead, the connection needs to be established by the user using the output device (screen and input devices). In this case, the pairing based on QR code might not be possible. Alternatively, the user can enter the XMPP user of the main device manually and some authentication/authorization procedure needs to be performed for security purposes (check whether the user is authorized to connect to the main device). As entering the XMPP user puts some effort on the user, an NFC tag on the main device could be read by an NFC reader in the output device instead. The proximity proved by the NFC communication could also avoid the need to another authentication/authorization procedure.

Extension to support headless main device

The main device could not only have no input device but even have no screen. In this case, screen mirroring is not possible and the screen of the output device would be used as the (current) primary screen of the main device. Connection setup would be initiated as described for main devices without input devices in the previous subsection.

The main device could be a wrist watch. Another option would be to have no physical main device at all but to have a virtual device running in the cloud.

Output device does not need to be local

The connection establishment using a link contained in a QR-code or NFC tag can be generalized. A link can be shared by arbitrary means, e.g. via email. Permissions whether a connection is granted or not need to be checked in the negotiation phase.

Multiple output devices can show the screen of the main device simultaneously

The screen content can be multicast to multiple output devices. Duplication can be done in the main device or in the backend. The latter is preferred as it is expected to be connected with more bandwidth.

Some Example Use Cases

- Use screen in hotel room (e.g. TV) in hotel room as output device for media on personal smartphone or tablet.
- Screen (of output device) in hotel lobby shows hotel ads. The user can attach to the output device and book services using the credentials on the main device (user's smartphone) while using the big touchscreen (of output device) as input device.
- Show a video stored on my smartphone on the computer or TV of a friend. Benefit: Easy, no complex configuration required.