Zero Configuration Networking

Unil | HEC | dop lab
Benoît Garbinato
distributed object programming lab

What is Zeroconf networking?

- Zero Configuration Networking (Zeroconf) is a set of standards that aim at automatically creating a usable IP network in the absence of dedicated servers or manual configuration
- The Zeroconf specification was initiated and driven by Apple, whose implementation is known as Bonjour (formerly RendezVous)
- For this, a Zeroconf solutions must be able to:
  - allocate IP addresses without a DHCP server
  - allocate IP Multicast addresses without a MADCAP server
  - translate names into IP addresses without a DNS server
  - find services without a directory server

> *A zeroconf protocol is able to operate correctly in the absence of configured information from either a user or infrastructure services such as conventional DHCP or DNS servers. Zeroconf protocols may use configured information, when it is available, but do not rely on it being present.*
>
> ***The Internet Engineering Task Force (IETF)***

Application examples

- Printer discovery in an unknown environment
- Mobile ad hoc multi-player games
- Self-configured real-time collaborative editing
- Media content sharing in a wireless network

Implementations

- There exists various implementations of the zeroconf approach, some of them following the IETF standards
  - Bonjour (formerly Rendezvous) by Apple
  - Avahi (open source) for Linux and BSD Unix

  } compatible with IETF's standards, based on mDNS

  - Windows CE 5.0 by Microsoft
  - Jini by Sun Microsystems

  } not compatible with IETF's standards

- In the following we focus on Apple's Bonjour, which is built into Mac OS X (both IPV4 and IPV6) and comes with iTunes on Microsoft Windows

Link to ubiquitous computing

- Ubiquitous computing environments can clearly benefit from MANETS, which require:
  - no infrastructure
  - no centralized servers
  - no network administrator
  - no static configuration or topology

- Zeroconf protocols provide a natural support to ubiquitous computing and MANETs

A rich zeroconf scenario

Ⓑ

4. **B** discovers **A**'s ability to be remotely controlled
5. **B** controls **A** music program remotely

Ⓓ

6. **D** advertises its printing services
7. **A** send print jobs to **D**, which prints them

Ⓐ

1. **A** advertises its multimedia content and it availability to be remotely controlled

Ⓒ

2. **C** discovers **A**'s video content
3. **A** streams video to **C**, which displays it on TV

What makes a network anyhow?

- A generic view
  - A unique address assignment scheme
  - A name-to-address resolution scheme
  - A service discovery scheme
- In traditional Internet/intranets, we have:
  - DHCP, DNS infrastructures
  - statically configured hosts
- In MANETs, we have none of that...

Unique address assignment

- Link-local address assignment in IPV4
  - relies on the 169.254.0.0/16 prefix, which corresponds to IP addresses in range [169.254.1.0, 169.254.254.255]
  - relies on random address selection
  - relies on ARP-based duplicate address discovery protocol
  - is described in RFC 3927
- Link-local address assignment in IPV6
  - relies on the FE80::/10 prefix (1111111010 in binary)
  - relies on a set of rules for selecting addresses (RFC 3484)
  - relies on a Duplicate Address Discovery protocol (RFC 4862)
  - is described in RFCs 4862, 4291 and 3484

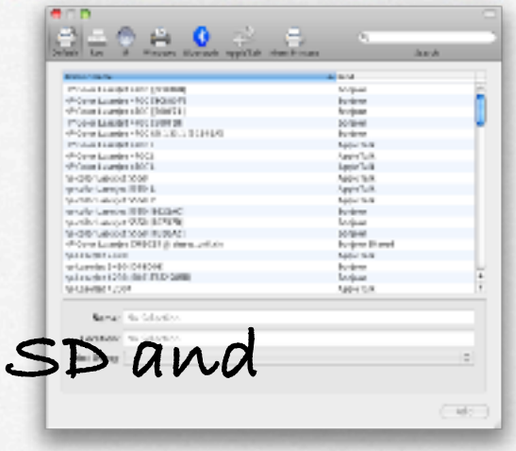
Name-to-address resolution (1)

- Name-to-address resolution in Bonjour is based on an implementation of the Multicast DNS standard (mDNS)
- The client multicasts an almost standard DNS query
- The target is a multicast address (group) on port 5353:
  - Multicast address for IPV4: 224.0.0.251
  - Multicast address for IPV6: FF02::FB
- The corresponding host, which is member of the multicast group, replies to that query
- Replies are multicast (all clients benefit from queries)

Name-to-address resolution (2)

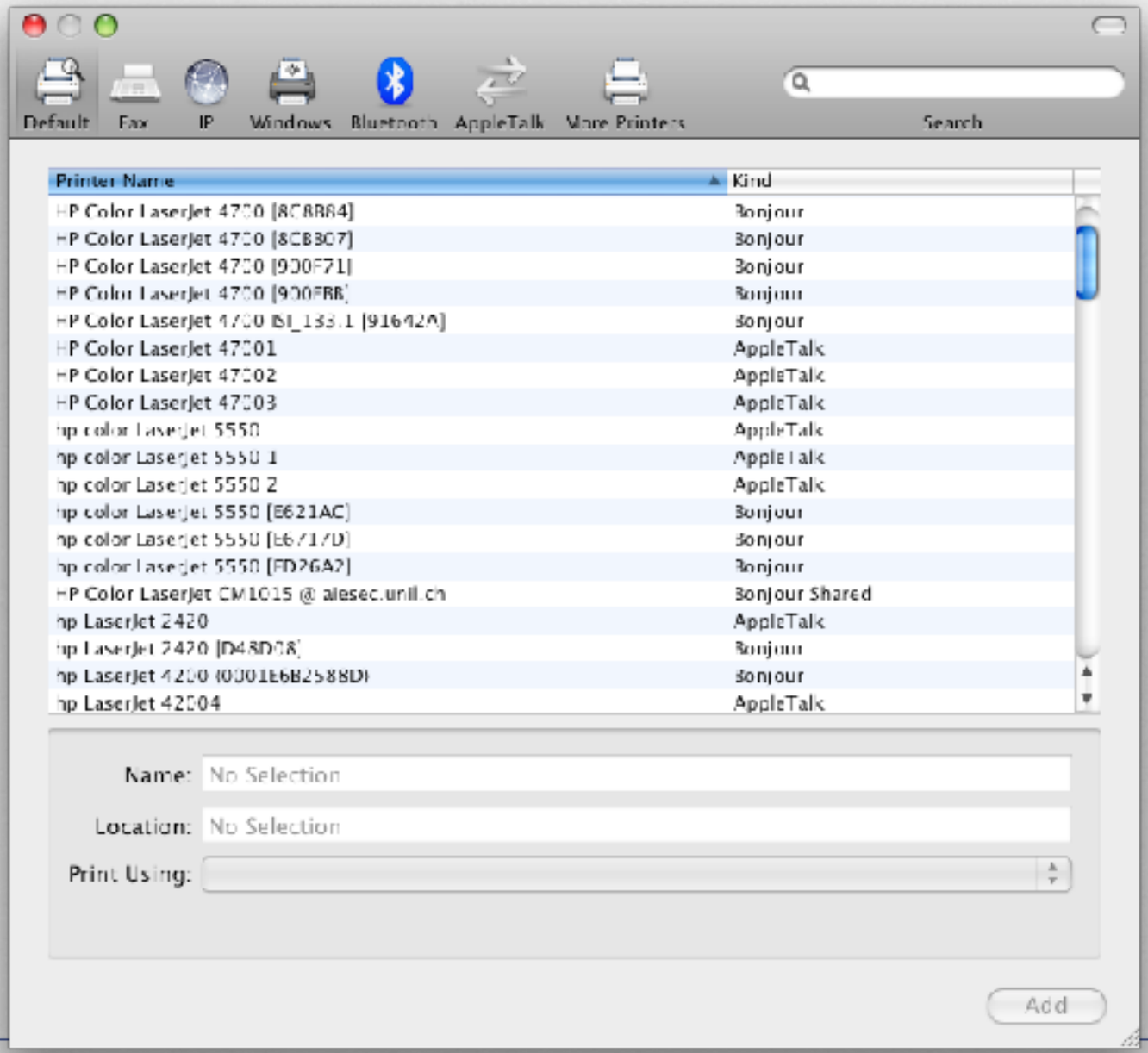
- To obtain a name, a host does the following:
  - it creates the name it wants to use
  - it issues a query to see whether there is a conflict
  - if it was the first to get the name, it wins
- In the case of race condition, the host with the lower address wins (possibly via negotiation)

Service discovery



- Service discovery is also based on mDNS, plus on DNS SD and DNS SRV records (RFC 2782).
- Protocol for service discovery:
  1. the server advertises a service type, e.g, «_ipp._tcp.mydomain.com»
  2. the client browses for services by querying for PTR records using the same service type, i.e., «_ipp._tcp.mydomain.com»
  3. the client receives a list of «[instance].[service].[domain]» PTR records, e.g., «HPColorLaserJet4700._ipp._tcp.mydomain.com»
     «TheBigBossPrinter._ipp._tcp.mydomain.com»
  4. this list is typically displayed to the user, who choses one instance
  5. the client resolves the chosen service, by issuing an SRV query
  6. the client receives a complete SRV record, containing all the necessary information to connect to the chosen service

Service discovery



dop lab

Advertising a service in Java

```
import com.apple.dnssd.DNSSD;
import com.apple.dnssd.DNSSDException;
import com.apple.dnssd.DNSSDRegistration;
import com.apple.dnssd.DNSSDService;
import com.apple.dnssd.RegisterListener;
import java.nio.channels.ServerSocketChannel;
```

```
public class ServiceAnnouncer implements RegisterListener {
    static final String serviceName = "bank";
    static final String serviceType = "trading";
    static final String serviceProtocol = "tcp";
    static final String registrationType = "_" + serviceType + "._" + serviceProtocol;
    private DNSSDRegistration serviceRecord;
    private int listeningPort;

    public void registerService(int port) {
        try {
            listeningPort = port;
            serviceRecord = DNSSD.register(serviceName, registrationType, listeningPort, this);
        } catch (DNSSDException e) {
            System.err.println("Unable to register the service: " + e.getMessage());
        }
    }
    public void unregisterService() {
        serviceRecord.stop();
    }
  public void serviceRegistered(DNSSDRegistration registration, int flags,
                String serviceName, String regType, String domain) {
            System.out.println("-> Service " + serviceName + " registered in domain " + domain);
        }
    }
    public void operationFailed(DNSSDService registration, int error) {
        System.err.println("-> Service registration failed");
    }
}
```

```
ServiceAnnouncer service = new ServiceAnnouncer();
service.registerService(port);
...
service.unregisterService();
```

Browsing (& finding) a service in Java

```
public static void main(String[] args) {
    try {
        ServiceBrowser browser= new ServiceBrowser();
        browser.startBrowsing();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(-1);
    }
}
```

```
import com.apple.dnssd.BrowseListener;
import com.apple.dnssd.DNSSD;
import com.apple.dnssd.DNSSDException;
import com.apple.dnssd.DNSSDService;

class ServiceBrowser implements BrowseListener {
    public void operationFailed(DNSSDService service, int errorCode) {
        System.out.println("Browse failed " + errorCode);
        System.exit(-1);
    }
    public void serviceFound(DNSSDService browser, int flags, int ifIndex,
            String name, String regType, String domain) {
        System.out.println("Service " + regType + " on " + name + " was found.");
        System.out.println("Interface is " + DNSSD.getNameForIfIndex(ifIndex));
    }
    public void serviceLost(DNSSDService browser, int flags, int ifIndex,
            String name, String regType, String domain) {
        System.out.println("Service " + regType + " on " + name + " was lost.");
        System.out.println("Interface is " + DNSSD.getNameForIfIndex(ifIndex));
    }
    public void startBrowsing() throws DNSSDException, InterruptedException {
        DNSSDService browsing = DNSSD.browse("trading._tcp", this);
        ...
        browsing.stop();
    }
}
```

Resolving a service in Java

```
public class ServiceResolver implements ResolveListener {
    SocketChannel channel;
    ...
    public void operationFailed(DNSSDService service, int errorCode) {
        System.out.println("Bonjour operation failed " + errorCode);
        System.exit(-1);
    }

    public void serviceResolved(DNSSDService resolver, int flags, int ifIndex,
            String fullName, String theHost, int thePort, TXTRecord txtRecord) {
        ByteBuffer buffer = ...;
        try {
            InetSocketAddress socketAddress = new InetSocketAddress(theHost, thePort);
            channel = SocketChannel.open(socketAddress);
            channel.write(buffer);
            ...
        } catch (Exception e) {
            e.printStackTrace();
        }

        resolver.stop();
    }
    public void startResolving(String name, String domain) {
        DNSSDService resolving = DNSSD.resolve(0, DNSSD.ALL_INTERFACES, name,
                                               "trading._tcp", domain, this);
        ...
        resolving.stop();
    }
}
```