

UDP Sockets: Peer to Peer Communication

By Anouar Moustarih, Quality Support Engineer, 4D Morocco

Technical Note 25-12

Table of Contents

Table of Contents	2
Abstract.....	3
Introduction	3
Prerequisites & Requirements.....	3
Prerequisites	3
Requirements	3
Peer Discovery Mechanism.....	4
Class Overview	4
Constructor.....	9
_initSockets	9
sendAnnouncement.....	9
sendProbe	9
onData	9
addOrRefreshPeer	10
removePeerById	10
prunePeers	10
stop	10
getPeers	10
Supporting Project Methods	10
Overview	10
RunPeerDiscovery.....	10
Util_formatLastSeen	11
Demo Application	12
Conclusion	12

Abstract

The User Datagram Protocol (UDP) serves as a lightweight, connectionless transport layer protocol in the TCP/IP suite, prioritizing speed and low overhead over reliability by omitting handshakes and retransmissions. UDP suits real-time applications where occasional packet loss is acceptable, such as video streaming or gaming, by allowing multicast and broadcast transmissions to multiple recipients efficiently. Peer-to-peer (P2P) systems built on UDP leverage these traits for decentralized discovery, where nodes broadcast presence messages to locate others without central coordination, reducing latency and infrastructure needs. Real-world use cases include file-sharing networks like BitTorrent for efficient content distribution, multiplayer games for quick player matching, IoT device coordination in smart homes for automatic service detection, and collaborative tools like video conferencing apps for local participant finding, all benefiting from UDP's simplicity in handling dynamic connections.

Introduction

This technical note details a comprehensive implementation of a peer-to-peer discovery mechanism, centered on the PeerDiscovery class and supporting methods for local private networks. The class utilizes 4D's UDPSocket and UDPEvent classes to handle broadcast announcements, probe responses, and event-driven data reception, maintaining a shared collection of peers with automatic expiration via TTL pruning. Key components include lifecycle management for initialization and shutdown, peer addition/refresh logic with shared object safety. Accompanied by the RunPeerDiscovery worker for continuous operation and Util_formatLastSeen for timestamp formatting, the system enables any 4D application to detect and track instances on the same network without servers or dependencies.

Prerequisites & Requirements

Prerequisites

Things that must be ready before launching the project

- 4D 20R10 or later is mandatory as UDPSocket and UDPEvent classes were introduced in 20R10.
- All application's instances must run on the same local network/subnet (broadcast reachability required)
- The chosen UDP discovery port (default 54321) must be allowed for both inbound and outbound traffic by any firewall, antivirus, or network policy on every machine
- No other process on the same machine may already bind to the discovery port (exclusive binding)
- macOS/Windows must allow the 4D application to open network sockets (standard user rights are sufficient)

Requirements

Conditions that must remain satisfied while the system is running

- UDP broadcast packets (destination 255.255.255.255 or the configured broadcast address) must reach all participating machines
- The discovery port must remain open and unblocked for the entire lifetime of the application

When these prerequisites and requirements are met, the Peer Discovery system starts instantly, and peers become visible within a few seconds. Any missing prerequisite or violated requirement results in silent failure or runtime errors.

Peer Discovery Mechanism

Class Overview

The PeerDiscovery class enables peer detection in local networks using UDP broadcasts. The class handles announcements, probes, and pruning automatically. The class operates by sending and receiving JSON-formatted announcements and probes. Peers appear in a shared collection for access across processes. The mechanism relies on UDP for low-overhead discovery. Broadcasts ensure local visibility without rendezvous servers. Probes maintain active peers. TTL prevents stale entries. Shared elements support preemptive execution. Functions interlink for seamless operation.

Properties define the class's state. The `listener` property holds a UDPSocket object for incoming data. The `sender` property manages outgoing broadcasts. The `discoveryPort` property sets the UDP port, defaulting to 54321. The `broadcastAddress` property targets "255.255.255.255" for local broadcasts. The `peers` property stores a shared collection of peer objects, each with id, name, ip, port, lastSeen, and info. The `myInfo` property contains the local peer's announcement data. The `peerTTL` property defines expiration in seconds, defaulting to 90. The `running` property flags active status. The `logText` property accumulates logs. The `probeInterval` property sets probe frequency in seconds, defaulting to 30.

```
property listener : 4D.UDPSocket
property sender : 4D.UDPSocket
property discoveryPort : Integer
property broadcastAddress : Text
property peers : Collection // collection of peer objects {id,name,ip,port,lastSeen,info}
property myInfo : Object // the object we announce about ourselves
property peerTTL : Integer // seconds after which a peer is considered gone
property running : Boolean
property logText : Text
property probeInterval : Integer
property closingSignal : 4D.Signal
```

```
shared Class constructor($options : Object)
  If (OB Is defined($options; "port"))
    This.discoveryPort:=OB Get($options; "port")
  Else
```

```

        This.discoveryPort:=54321
    End if
    If (OB Is defined($options; "broadcastAddress"))
        This.broadcastAddress:=OB Get($options; "broadcastAddress")
    Else
        This.broadcastAddress:="255.255.255.255"
    End if
    If (OB Is defined($options; "ttl"))
        This.peerTTL:=OB Get($options; "ttl")
    Else
        This.peerTTL:=90
    End if
    If (OB Is defined($options; "probeInterval"))
        This.probeInterval:=OB Get($options; "probeInterval")
    Else
        This.probeInterval:=30
    End if
    var $id : Text
    If (OB Is defined($options; "id"))
        $id:=OB Get($options; "id")
    Else
        $id:=Generate UUID()
    End if
    var $name : Text
    If (OB Is defined($options; "name"))
        $name:=OB Get($options; "name")
    Else
        $name:=System info.machineName
    End if
    var $servicePort : Integer
    If (OB Is defined($options; "servicePort"))
        $servicePort:=OB Get($options; "servicePort")
    Else
        $servicePort:=0
    End if
    var $version : Text
    If (OB Is defined($options; "version"))
        $version:=OB Get($options; "version")
    Else
        $version:=System info.osVersion
    End if
    This.myInfo:=New shared object("type"; "ANNOUNCE"; "id"; $id; "name"; $name; "port"; $servicePort;
"version"; $version)
    This.peers:=New shared collection()
    This.running:=True
    This.logText:=""
    This.closingSignal:=New signal()
    var $socketHolder : Object:=New shared object()
    Use ($socketHolder)
        $socketHolder.listener:=Null
        $socketHolder.sender:=Null
        $socketHolder.endOfInitialization:=New signal
    End use

```

```

CALL WORKER("p2pworker"; Formula($1._initSockets($2)); This; $socketHolder)
$socketHolder.endOfInitialization.wait()
This.listener:=$socketHolder.listener
This.sender:=$socketHolder.sender
Function _initSockets($socketHolder : Object)
    Use ($socketHolder)
        $socketHolder.listener:=4D.UDPSocket.new(This.discoveryPort; This)
        $socketHolder.sender:=4D.UDPSocket.new(0; This)

    End use
    $socketHolder.endOfInitialization.trigger()
Function sendAnnouncement()
    var $json : Text
    $json:=JSON Stringify(This.myInfo)
    var $blob : Blob
    TEXT TO BLOB($json; $blob; UTF8 text without length)
    Use (This)
        This.logText:=This.logText+"["+Time string(Current time)+"] Sent ANNOUNCE\n"

    End use
    This.sender.send($blob; This.broadcastAddress; This.discoveryPort)
Function sendProbe()
    var $probe:=New shared object("type"; "PROBE"; "id"; This.myInfo.id)
    var $json : Text
    $json:=JSON Stringify($probe)
    var $blob : Blob
    TEXT TO BLOB($json; $blob; UTF8 text without length)
    Use (This)
        This.logText:=This.logText+"["+Time string(Current time)+"] Sent PROBE\n"

    End use
    This.sender.send($blob; This.broadcastAddress; This.discoveryPort)
Function onData($socket : 4D.UDPSocket; $event : 4D.UDPEvent)
    var $text : Text
    $text:=BLOB to text($event.data; UTF8 text without length)
    var $obj : Object
    $obj:=New object
    $obj:=JSON Parse($text)
    If (Type($obj)=Is object)
        var $msgType : Text
        If (OB Is defined($obj; "type"))
            $msgType:=OB Get($obj; "type")
        Else
            $msgType:=""
        End if
        var $senderId : Text
        If (OB Is defined($obj; "id"))
            $senderId:=OB Get($obj; "id")
            Use (This)
                This.logText:=This.logText+"["+Time string(Current time)+"] Received
                "+$msgType+" from "+$event.address+":."+String($event.port)+"\n"
            End use
        Else
            $senderId:=""
        End if
        If ($senderId=This.myInfo.id)

```

```

        return
    End if
    If ($msgType="ANNOUNCE")
        This.addOrRefreshPeer($obj; $event.address; $event.port)
    Else
        If ($msgType="PROBE")
            var $json : Text
            $json:=JSON Stringify(This.myInfo)
            var $blob : Blob
            TEXT TO BLOB($json; $blob; UTF8 text without length)
            This.sender.send($blob; $event.address; $event.port)
        Else
            If ($msgType="BYE")
                This.removePeerById($senderId)
            End if
        End if
    End if
End if

Function addOrRefreshPeer($peerObj : Object; $ip : Text; $port : Integer)
    var $id : Text
    If (OB Is defined($peerObj; "id"))
        $id:=OB Get($peerObj; "id")
    Else
        $id:=""
    End if
    If ($id="")
        return
    End if
    var $idx : Integer
    $idx:=-1
    var $i : Integer
    For ($i; 0; This.peers.length-1)
        If (This.peers[$i].id=$id)
            $idx:=$i
            break
        End if
    End for
    var $peerName : Text
    If (OB Is defined($peerObj; "name"))
        $peerName:=OB Get($peerObj; "name")
    Else
        $peerName:=""
    End if
    Use (This.peers)
        var $entry : Object
        $entry:=New shared object("id"; $id; "name"; $peerName; "ip"; $ip; "port"; $peerObj.port;
"lastSeen"; Milliseconds)
        If ($idx#-1)
            This.peers[$idx]:=$entry
        Else
            This.peers.push($entry)
        End if
    var $info : Object

```

```

        $info:=New shared object("type"; $peerObj.type; "id"; $id; "name"; $peerName; "port";
$peerObj.port; "version"; $peerObj.version)
        $entry.info:=$info
    End use
Function removePeerById($id : Text)
    If ($id="")
        return
    End if
    var $i : Integer
    Use (This.peers)
        For ($i; 0; This.peers.length-1)
            If (This.peers[$i].id=$id)
                This.peers.remove($i; 1)
                break
            End if
        End for
    End use

Function prunePeers()
    var $now : Integer
    $now:=Milliseconds
    var $stillAlive : Collection
    $stillAlive:=New shared collection
    Use (This.peers)
        var $i : Integer
        For ($i; 0; This.peers.length-1)
            var $diff : Real
            $diff:=( $now-This.peers[$i].lastSeen)/1000
            If ($diff<This.peerTTL)
                var $old : Object
                $old:=This.peers[$i]

                var $newInfo : Object
                $newInfo:=New shared object("type"; $old.info.type; "id"; $old.id; "name";
$old.name; "port"; $old.port; "version"; $old.info.version)

                var $newEntry : Object
                $newEntry:=New shared object("id"; $old.id; "name"; $old.name; "ip"; $old.ip;
"port"; $old.port; "lastSeen"; $old.lastSeen; "info"; $newInfo)
                $stillAlive.push(Ob Copy($newEntry; ck shared; $stillAlive))
            End if
        End for
    Use (This)
        This.peers:=$stillAlive.copy(ck shared; This)
    End use
End use

Function stop()
    var $bye:=New shared object("type"; "BYE"; "id"; This.myInfo.id)
    var $json : Text
    $json:=JSON Stringify($bye)
    var $blob : Blob
    TEXT TO BLOB($json; $blob; UTF8 text without length)
    If (This.sender#Null)

```



```

        This.sender.send($blob; This.broadcastAddress; This.discoveryPort)
    Use (This)
        This.logText:=This.logText+"["+String(Current time)+"] Sent BYE\n"
    End use
Else
    Use (This)
        This.logText:=This.logText+"["+String(Current time)+"] Skipped BYE send (sender not
initialized)\n"
    End use
End if
This.closingSignal:=New signal()

Function getPeers() : Collection
    return This.peers.copy(ck shared)

```

Constructor

The shared constructor accepts an options object. The constructor processes port, broadcastAddress, ttl, and probeInterval from options. Default values apply if options lack keys. The constructor generates a UUID for id if absent. Machine name fills name if unspecified. Service port defaults to 0. OS version sets version if missing. The constructor builds myInfo as a shared object with type "ANNOUNCE". Peers initializes as a shared collection. Running sets to True. LogText starts empty. A socketHolder shared object prepares for worker. The constructor calls a worker for socket initialization. EndOfInitialization signal waits. Listener and sender assign from socketHolder.

_initSockets

The _initSockets function creates sockets. The function uses socketHolder. Listener binds to discoveryPort. Sender uses port 0. Signal triggers completion.

sendAnnouncement

The sendAnnouncement function broadcasts myInfo. JSON stringifies myInfo. Text converts to blob. Log appends "Sent ANNOUNCE". Sender transmits to broadcastAddress on discoveryPort.

sendProbe

The sendProbe function sends probe messages. Probe object holds type "PROBE" and id. JSON stringifies probe. Blob creates from text. Log adds "Sent PROBE". Sender broadcasts to address and port.

onData

The onData function processes UDP events. Blob converts to text. JSON parses into obj. Type checks for object. MsgType extracts from obj.type. SenderId pulls from obj.id. Log records receipt with address and port. Function returns if senderId matches local id. For

"ANNOUNCE", addOrRefreshPeer calls with obj, address, port. For "PROBE", myInfo sends back to sender. For "BYE", removePeerById invokes with senderId.

addOrRefreshPeer

The addOrRefreshPeer function updates peers. Id extracts from peerObj. Function returns if id empty. Index searches for matching id in peers. PeerName pulls from peerObj.name. Use block locks peers. Entry creates as shared object with id, name, ip, peerObj.port, lastSeen as Milliseconds. Entry assigns or pushes based on index. Info recreates as shared with type, id, name, port, version. Info assigns to entry.info.

removePeerById

The removePeerById function deletes peers. Function returns if id empty. Use block locks peers. Loop finds matching id. Remove calls with index.

prunePeers

The prunePeers function cleans stale peers. Now sets to Milliseconds. StillAlive initializes as shared collection. Use block locks peers. Loop calculates diff as (now - lastSeen)/1000. If diff below peerTTL, old copies peer. NewInfo recreates shared from old.info. NewEntry builds shared with old data and newInfo. StillAlive pushes copy of newEntry. Peers assigns copy of stillAlive.

stop

The stop function ends discovery. Bye object holds type "BYE" and id. JSON stringifies bye. Blob creates. Sender transmits if not null. Log adds "Sent BYE" or skip message. The background worker will stop sending probes and will close sockets

getPeers

The getPeers function returns peers copy .copy uses ck shared .

Supporting Project Methods

Overview

Two dedicated project methods complete the peer discovery system. RunPeerDiscovery operates as the background worker that keeps the discovery alive. Util_formatLastSeen transforms raw timestamps into readable strings for the user interface. These methods work outside the PeerDiscovery class yet remain essential. RunPeerDiscovery is executed once in a worker. Util_formatLastSeen is called from listbox column expressions, leveraging automatic updates and clear "time ago" displays.

RunPeerDiscovery

```

#DECLARE($options : Object)

var $discovery : cs.PeerDiscovery
$discovery:=cs.PeerDiscovery.new($options)

Use (Storage)
    Storage.discovery:=$discovery // Share instance
    Storage.peers:=New shared collection
    Storage.logText:=New shared object("value"; "")
End use

While (Not($discovery.closingSignal.wait(5)))

    $discovery.sendProbe()
    $discovery.prunePeers()
    Use (Storage)
        Storage.peers:=$discovery.getPeers() //.copy(ck shared)
        Use (Storage.logText)
            Storage.logText.value:=$discovery.logText
        End use
    End use

End while

Use (Storage)
    Storage.discovery:=Null
    Storage.peers:=New shared collection
    Storage.logText:=New shared object("value"; "")
End use

```

The RunPeerDiscovery method accepts an options object as input. The method creates a new signal object. The signal assigns to options.signal for coordination. A PeerDiscovery instance initializes with the options. Shared storage updates with the discovery instance. Storage.peers sets to a new shared collection. Storage.logText initializes as a shared object with empty value.

A loop runs while the signal remains unsignaled. Probes send through the discovery instance. Pruning removes expired peers. Storage.peers updates with a copy from getPeers. LogText.value assigns the discovery's logText within nested Use blocks. Process delays for 30 seconds using 5*60 ticks.

Upon closing signal, the method stops the discovery instance. Storage clears by nulling discovery and resetting peers and logText.

Util_formatLastSeen

```

#DECLARE($lastSeen : Integer)->$formatted : Text

var $diff : Real:=(Milliseconds-$lastSeen)/1000 // Seconds elapsed

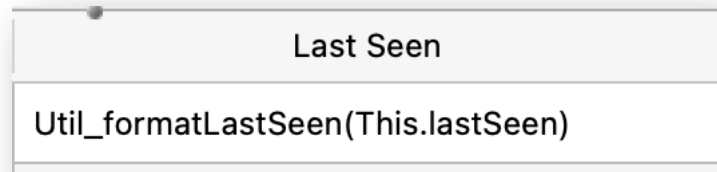
```

```

Case of
    : ($diff<60)
        $formatted:=String(Round($diff; 1))+ " seconds ago"
    : ($diff<3600)
        $formatted:=String($diff\60)+ " minutes ago"
    : ($diff<86400)
        $formatted:=String($diff\3600)+ " hours ago"
End case

```

The Util_formatLastSeen method declares lastSeen as Integer input and formatted as Text output. Diff calculates as milliseconds minus lastSeen divided by 1000 seconds.



Demo Application

The application must be launched with version 20R10 or higher.

Upon initialization, the probing workflow starts automatically, and each probe event is written to the application log.

Activation of the Announce button triggers the generation of an announcement message. This message is handled by the same logging mechanism used for probe events.

The Local Info field exposes the attributes of the local peer. These attributes correspond to the underlying system information of the host machine acting as the peer.

The peer list undergoes continuous synchronization. Stale peers are pruned, and newly discovered peers are appended. Both operations are executed in the background and are fully orchestrated by the PeerDiscovery class.

The Close button terminates the peer-discovery sequence on the local network. The peer executing this action is removed from active discovery.

Conclusion

The PeerDiscovery class, delivers a complete peer-to-peer discovery solution entirely within native 4D. By harnessing the power of UDPSocket and UDPEvent classes, the system achieves true asynchronous, event-driven UDP communication.

The implementation requires no external libraries, no rendezvous server, and no manual configuration. Once launched, the 4D application instantly gains real-time awareness of all other running instances on the same local network. Automatic TTL-based pruning keeps the

peer list accurate, graceful BYE messages prevent ghost entries, and the provided formatting method ensures clean presentation in user interfaces.

This solution proves particularly valuable for distributed tools, collaborative desktop applications, local synchronization services, and any scenario requiring rapid, serverless instance coordination.