DirectPlay Game Query

Star Trek: Armada could be played basically via one of the following network means:

1. TCP/IP LAN play,
2. IPX LAN play and
3. WON internet play.

The actual game play is facilitated by DirectPlay connections. The queries for open games are slightly different for the above mentioned types of connection/game set-up. But in essence DirectPlay is always involved. As WON worked differently in terms of how to gain a list of open games, it will not be described here. But for the LAN play options, see below. They all do DirectPlay game queries, giving them the information of where a game is open, and what properties it does have (e.g. game name or map being used).

TCP/IP vs. IPX

Basically it is not really relevant, how exactly the TCP/IP connection or the IPX connection is established. The actual contents sent by the game must of course be transported properly to the other peers. But whether it is a native IPX connection or an RFC 1234 IPX server, are details the game will know nothing about. Same goes for TCP via some VPN solution. It is just important to note, that when doing an analysis of network traffic, one must of course keep in mind, that overhead from other protocols (e.g. RFC 1234 UDP headers and IPX headers) is to be excluded from any analysis for DirectPlay game queries.

When receiving the TCP packages, the following socket header is considered to be part of the DirectPlay header by *WireShark*. Considering that it is not part of the IPX packages, it stands to reason, that this based on a wrong assumption that it is part of the DirectPlay contents. It must be skipped to get to the actual DirectPlay contents.

Offset	Description
00/00	Size of the entire part, including the DirectPlay contents (0x8e 0x00)
02/02	Token (0xb0 0xfa)
04/04	AF_NET (0x02 0x00)
06/06	Port for the UDP session, 2 Bytes
08/08	IP (0x00 0x00 0x00 0x00)
12/0c	Padding (0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00)

So in total a length of 20 bytes can be skipped, to get to the DirectPlay contents. The offsets below are all based on the assumption, that this socket header is already stripped from the data.

The DirectPlay Analysis

Method

For the analysis of the game's sent and received data Wireshark was used on a LAN, once with direct TCP/IP packages used and once with an RFC 1234 UDP-IPX-server in-between the peers. This already requires to carefully look at the results. Apparently Wireshark does not know about RFC 1234 wrapped IPX messages. It does recognize the UDP connection and also that there is IPX payload inside of the UDP datagram. But it does not recognize the DirectPlay payload inside the IPX package as such. However, when connecting directly via TCP/IP it does indeed recognize DirectPlay contents for what they are and gives a detailed analysis of the different parts of each DirectPlay package, like headers or what kind of fields and flags are being used.

When looking at the packages sent back and forth between a peer having an open game lobby, and a peer requesting open lobbies via broadcast from everyone else on the same subnet, the contents are fairly the same for TCP/IP and IPX. The network headers are different of course (the IPX header being removed leaves the very same contents, as are present via TCP/IP, while TCP/IP seems to add socket headers before those).

So for the analysis of IPX sent DirectPlay the TCP/IP representation was taken as a base of reference. Headers that have nothing to do with DirectPlay were removed, leaving only the following contents:

Query Contents

The open game query is actually rather short, once headers of TCP or IPX are removed:

Group	Offset (Dec/Hex))	Description
Header	00/00	The word *play* (0x70 0x6c 0x61 0x79)
Header	04/04	Query Enum 2, presumably marking the package as a sessions request (0x02 0x00)
Header	06/06	DirectPlay Version 9 (0x0e 0x00)
Data	08/08	DirectPlay Game GUID, this identifies Armada games (0x01 0x38 0xf9 0x76 0x40 0x93 0xd2 0x11 0xae 0x34 0x00 0x60 0x08 0x95 0xc7 0x79)
Data	24/18	PW Offset (0x00 0x00 0x00 0x00)
Data	28/1c	Sess.Flags (0x01 0x00 0x00 0x00)

Session Flags (Sess.Flags) are a bit field with the following meanings:

Bit Position	Description
1-25	Nothing?
26	Session Requiring a Password
27-30	Nothing?
31	All Sessions
32	Joinable Sessions

The game queries all games, not filtering for anything. The filtering happens on the client side. So the in-game selection field does not control which contents come from the peers having a session opened.

These messages are sent to the broadcast address. For TCP/IP that is 255.255.255.255, for IPX it is ff:ff:ff:ff:ff:ff. Note: In case of the TCP/IP broadcast, the game apparently only uses the primary

network card (whose metric is the lowest). Meaning, if you are connected to multiple networks, only on of them is actually considered. If that is some VPN solution's network card, then an actual LAN game cannot be found in this fashion.

Answer Contents

General DirectPlay Structure

The answer to such a request is relatively long, but mostly because the payload needs actual space:

Group	Offset (Dec/Hex))	Description
Header	00/00	The word *play* (0x70 0x6c 0x61 0x79)
Header	04/04	Query Enum 2, presumably marking the package as a sessions request (0x02 0x00)
Header	06/06	DirectPlay Version 9 (0x0e 0x00)
Data	08/08	Length
Data	12/0c	Session Description Flags
Data	16/10	Instance GUID
Data	32/20	Game GUID (0x01 0x38 0xf9 0x76 0x40 0x93 0xd2 0x11 0xae 0x34 0x00 0x60 0x08 0x95 0xc7 0x79)
Data	48/30	Maximum number of players (for the actual game this is always 8, so probably ignored) (0x00 0x00 0x00 0x00)
Data	52/34	Number of currently connected players (At least 1, 8 tops)
Data	56/38	Name pointer
Data	60/3c	Password pointer
Data	64/40	Reserved 1 (?)
Data	68/44	Reserved 2 (0x00 0x00 0x00 0x00)
Data	72/48	Description 1 (?)
Data	76/4c	Description 2 (map name block 1)
Data	80/50	Description 3 (map name block 2)
Data	84/54	Description 4 (map name block 3)
Data	88/58	Name Offset, the location inside the DirectPlay content, that gives the lobby name (0x5c)
Data	92/5c	Lobby name, 30 bytes

Session Description Flags are a bit field of length 32 bits/4 bytes with the following meaning:

Bit Position	Description
1-14	Nothing
15	No session Description
16	Acquire Voice
17	Optimize for Latency
18	Preserve Order
19	Use Reliable Protocol
20	Get Server Player Only
21	Route Via Game Host
22	Password Required

Bit Position	Description
23	Private Session
24	Use Authentication
25	No Player Updates
26	Use Ping
27	Can Join
28	Ignored
29	Short Player Message
30	Migrate Host Flag
31	Unused
32	No Create Players

Note: The *Session Description Flags* like *Can Join* have nothing to do with the actual Armada information about the match, e.g. whether the game is closed off, or not. Some of that information can be found in the *Description 1* block.

Description 1

Reserved 1 and Description 1 seem to change. The rest is (aside from the obvious GUIDs) static. *Description 1* holds at least the following information. The offset values are 0-based in relation to the entire DirectPlay block (first byte has number 0, values only start at 72). The bits are 1 based.

Byte Dec	Byte Hex	Bit	Description
74	4a	6	Is a match already ongoing (has started)? 1 means yes, 0 means no.
74	4a	7	Has the match been closed (no joining possible)? 1 means yes, 0 means no.
74	4a	3	Has the match been protected by a password? 1 means yes, 0 means no.

[**Star Trek: Armada**] [Game Play] [Technical Support] [Trivia]

[Units] [Hero Ships] [Campaign Ships] [Ship Types] [Stations] [Station Types] [Special Weapons] [Factions] [Modding]

[Back to the Games Database]