SDCP - Simple Distributed Computing Protocol

Glenn Larsson [ichinin@SUESPAMMERS.org]

Abstract:

SDCP is a lightweight but practical protocol for scheduling jobs, sending and recieveing data in a distributed environment. "Distributed" is multiple nodes (computers) talking to a central node that hands out jobs.

Glenn Larsson
[February, 2001]

Table of Contence:

SECTION 1

1.1 - Introduction:

SDCP is a *distributed computing protocol* for scheduling jobs from a central *node* (server) to multiple concurrent nodes (Clients).

The protocol is designed to be as "braindead" (non complex) and simple to implement as possible. (Since i am a sparetime researcher, i do not have time to create anything but a functional communications protocol.)

This protocol also assumes that thre reader are familliar with such technologies such as IP protocols, some cryptology and distributed computing.

1.2 - Purpose:

SDCP is fulfilling a (personal) need for a "piece of cake" protocol that allow you to use whatever hardware and software available to you as a node in a distributed system. Beowulf is not an option in my case; since i have access to mainly wintel systems.

1.3 - Terminology:

Client:

A node (See "Node") that do jobs and talks to the central node (See "Server")

Cryptografy:

See "Encryption".

Distributed Computer:

A multi node computer that can do many tasks in parallel, this protocol refers to multipple computers that are connected via network or other comminications link. It does not refer to neither DCOM OR a multiprocessor environment (such as a system with a shared memory.)

Distributed computing:

See "Distributed Computer".

Encryption:

Means of data hiding.(See "One Way Hash Algorithm")

Encryption key:

The "Master secret" that allow encryption to encrypt (= hide) and decrypt (= make readable again) data.

Hash Algorithm:

See "One Way Hash Algorithm".

Key Agreement:

A way of negotiating an encryption key (See "Encryption", "Encryption Key") for a specific session (See "Session").

MAC:

Message Authentication Code. See "One Way Hash Algorithm".

MD:

Message Digest. A Hash for validating data integrity.
(See "One Way Hash Algorithm")

Node:

A single computer that either hands out jobs (See "Server") or do jobs (See "Client") In other words: a standard PC with a network card or other communications interface. See "Distributed Computer".

One Way Hash Algorithm:

A "Data Fingerprint"; The unique characteristics of the data represented by a "Hash" ~= a hexadecimal representation of the data. Change 1 bit and approximately 50% of the output bits will change, giving you a totally new signature of the data. It is impossible to reverse (hashed) data into it's original data, one can make guesses, but that’s another discussion.

One Way Hash algorithms are used for digital signatures, Message Authentication Codes (MAC), Intrusion Detection Systems, Pseudorandom Number Generators (PRNG's) and various encryption key agreement protocols.

OSI:

Open System Interconnect / Open System Interface
(Depending on who you talk to...)

A specification that allow a multitude of hardware and software to talk to eachother.

Server:

The central node that hands out jobs.

Session:

When 2 computers talk to each other. Includes handshaking (initialisation) and termination (finalisation)

TCP:

A way of communicating using Sockets (See "Socket").

Socket:

A way of sending data over a network (usually the internet)

1.4 - Explainations:

- All the examples are specific for the IP protocol suite, since i'll be playing with computers and their associated network gear - i'll be using using a plain, default, vanilla, standard TCP Socket.

- Each item is described in this way:

- **Prototype:**
- **Example:**
- **Effect:**

Prototype - is a brief explaination of HOW the command and it's data should be assembled before it's sent to the server.

Example - is an example of what the data could look like.

Effect - explains what this particular method or command do.

- The " (Quote) characters, should be stripped from the left and the right of the examples, since every command and reply begin with a word + ":" (Colon). and ends with a line terminator; CRLF (Chr 13 and Chr 10)

SECTION 2 - Client commands sent to the Server

2.1 - Requesting a new job

When finished with a job, or when just started up, the client needs a job.

Prototype:

"JOBREQUEST:INSTALLED_PLUGINS|SIGNATURE" + CRLF

Example:

"JOBREQUEST:PLUGIN1|PLUGIN2|SIGNATURE"

Effect:

The client apply for a job from the server.

2.2 - Finalising a job (Successful, result is sent to server)

The scheduled job was executed succesfully. Send the result to the server

Prototype:

"FINALISE:JOB_ID|DATA|SIGNATURE" + CRLF

Example(s):

"FINALISE:1234|X=11,Y=23|SIGNATURE"
"FINALISE:1234|UNSUCCESSFUL|SIGNATURE"

Effect:

This removes the job from the cue on the server AND logs/presents the data in some (unspecified) way on the server.

2.3 – Indicating a job failure (Job unsuccessful, no data is sent to server)

The scheduled job was unsuccesfull. No data need to be returned. (Unable to execute JOBNODE or other failure)

Prototype:

"FAILURE:JOB_ID|SIGNATURE" + CRLF

Example:

"FAILURE:1234|SIGNATURE"

Effect:

This reschedules the job to another client.

2.4 - Indicating an Initial connection

When the client first connect, it will need to identify that it never have connected before, if the server should fail to identify that that particular node have indeed never have connected before.

(Suppose you reinstall the computer and give it the same config...)

Prototype:

"INITIAL:SIGNATURE" + CRLF

Example:

"INITIAL:SIGNATURE"

Effect:

The client tells the main node that "Hi - I'm the new guy!"

2.5 - Resynchronisation

Whenever a node looses the job it's doing, by unintended interference or by a BDP (Brain Dead Programmer) we would need the ability to say -"Huh?".

Prototype:

"REFRESH:SIGNATURE" + CRLF

Example:

"REFRESH:SIGNATURE"

Effect:

The previous response sent from the server is re-sent to the client.

2.6 – Reply to a sysinfo request

Direct reply format for a ”SYSINFOREQUEST” server command

Prototype:

"SYSINFOREPLY:DATA|SIGNATURE" + CRLF

Example:

"SYSINFOREPLY:P3-700,256MB RAM,40GB HDD|SIGNATURE"

Effect:

Returns the data requested by the server.

SECTION 3 - Server replies sent to the Client

3.1 - Sending a job to the client

The server reply with the job to do.

Prototype:

"JOB:JOBNODE|JOB_ID|DATA|SIGNATURE" + CRLF

Example:

"JOB:3|1234|PARAMETER_A=1,PARAMETER_B=2|SIGNATURE"

Effect:

A job is sent to the client, indicating that it should use plugin 3 (Jobnode 3.)

3.2 - Sending a new plugin to the client

Distribution of jobs is done by sending executables to the client.

Prototype:

"PLUGIN:JOBNODE|HEXADECIMAL_DATA|SIGNATURE" + CRLF

Example:

"PLUGIN:2|FE7A62A92...F78A0000|SIGNATURE"

Effect:

The server sends a binary file in hexadecimal format, in this example it is recieved by the current client and saved locally as "CLIENT2.EXE" (The JOBNODE is determined by the server).

3.3 - Sleep mode (There is nothing to do)

Sometimes, there are no pending jobs on the main node or the server don’t have enough sockets loaded to process the incomming request. This simply tells the client to sleep for a while and come back later.

Prototype:

"SLEEP:SECONDS|SIGNATURE" + CRLF

Example:

"SLEEP:30|SIGNATURE"

Effect:

Tells the client to sleep for N seconds; in the example we choosed 30 seconds. When N seconds have passed the client will connect to the main node and re-request a job.

3.4 - Quit

All jobs are done, we need a way to tell the client to shut itself down.

Prototype:

”QUIT:SIGNATURE" + CRLF

Example:

"QUIT:SIGNATURE"

Effect:

The client terminates itself.

3.5 - Sysinfo

Request that the client should identify itself, run a simple benchmark
On the systems (i.e. n number of multiplications, file read/write operations)

Prototype:

"SYSINFOREQUEST:SIGNATURE" + CRLF

Example:

"SYSINFOREQUEST:SIGNATURE"

Effect:

The server tells the client to give sysinfo and benchmark info.
(Sysinfo = Processor ID, Speed in Mhz,. Operating system)

SECTION 4 - Appendix

4.1 - SDCP in relation to the OSI model

Layer 7 - Application

- SDCP Provides client and server nodes with an interface to send and recieve data with.

Layer 6 - Presenation

- SDCP Translates and parses Client requests.
- SDCP Translates and parses Server response.

Layer 5 - Session

- SDCP controls and if necessary terminates the connection between the client and the server.

Layer 4 - Transport

SDCP utilises Message Authentication Codes and Signatures to provide Client and Server with error free data.

Layer 3 - Network

Nothing implemented at this level.

Layer 2 - Data Link

Nothing implemented at this level.

Layer 1 - Physical

Nothing implemented at this level.

4.2 - Example sessions using SDCP.

To be remade.

4.3 - Generics.

The protocol is correcting using MAC's, Signature or simple message digests, i suggest that, depending on your flavour of errorchecking, you choose a good hash algorithm (MD5 or SHA-1) to provide good data integrity for the sessions. (CRC32 will do fine though...)

The SIGNATURE field represented in most querys/replies are not mandatory, but you SHOULD use it.

If you want to be sure that the data can be validated and is unchanged, select a MAC. If the data is confidential, encrypt.

4.4 - References.

None.

Glenn Larsson
2001 March 6