Class ControllerServlet

Java servlet which controls the application.

Variables:

ServletContext ctx;

Holds the Servlet Context object.

URL urlNDL;

Path to the NDL-file.

URL urlWeight;

Path to the file which contains all weights.

NetworkDescription network;

Contains the complete SURFnet infrastructure.

List<String> sites;

Used by JSP files for displaying the lists of sites.

List<Interface> endpoints;

Used by JSP files for displaying the lists of endpoints.

List<Device> devices;

Used by JSP files for displaying the lists of devices.

List<Interface> interfaces;

Used by JSP files for displaying the lists of interfaces.

int *HOP_WEIGHT*;

Cost of a hop.

int *LOAD_WEIGHT*;

Cost of 1% load of an interface.

int *SHARED_DEVICE_WEIGHT*;

Weight of primary and backup path for sharing a device.

int *SHARED_RISKGROUP_WEIGHT*;

Weight of primary and backup path for sharing a riskgroup.

int *SHARED_LINK_WEIGHT*;

Weight of primary and backup path for sharing a link.

Methods:

void RefreshNetwork()

When called, the variable "network" gets refreshed.

void init(ServletConfig servletConfig) throws ServletException

Init method is overridden. In the new init two timers are set. If the NDL-file or the weight-file is updated then RefreshNetwork is called.

void doGet(HttpServletRequest request, HttpServletResponse response)

void doPost(HttpServletRequest request, HttpServletResponse response)

Responsible for interpreting the users requests and for returning the requested data.

String PrintFreeTimeslots(Interface ifc, int capacitysts1)

Prints the first “capacitysts1” free timeslots of interface “ifc”.

String PrintResults(Calculator cal, List<List<Interface>> paths, List<List<Interface>> optimalpath, int capacity, boolean printtimeslots)

Returns the results of the calculated path and optimal with the free timeslots in HTML code.

boolean StorePathInSession(HttpServletRequest request, List<List<Interface>> path, List<List<List<Integer>>> timeslots, Device viadev, Interface viaifc1, Interface viaifc2, String notviasite, Device notviadev, Interface notviaifc1, Interface notviaifc2, int capacity)

If a successful path is calculated all information is stored in the session of the user. With this information the path can be recalculated if the user wants to store the found path. If the recalculated path is equal to the previous found path then the path is stored otherwise some changes had been made to the network and the previous found path is not available anymore.

void CalculatePath(HttpServletRequest request)

Creates the “network” and puts it in the Calculator Class to calculate the optimal path.

void CalculateCurrentLightpath(HttpServletRequest request)

Give information about an active lightpath in the SURFnet network.

boolean SavePath(HttpServletRequest request)

Stores the found path to the MySQL database.

The methods goSingle, goCurrentLightpaths, goRanking, goHotspots, goRiskgroupsPerInterface shows the user the different menu items.

Class Calculator

NetworkDescription network;

A NetworkDescription object on which the calculations will be made.

List<List<Interface>> iterationresultfree;

result of SRLG-exclusion

List<List<Interface>> iterationresultnotfree;

primary path = primary path iterationresultfree, backup path is searched to be disjoint of primary path

List<Interface> shortestPath;

current shortest path of Dijkstra algorithm

int shortestPathCost;

cost of current shortest path of Dijkstra algorithm

Set<String> shortestPathWithout;

primary path was searched without these riskgroups

int capacity;

capacity of the searched path.

List<List<Interface>> explore = new ArrayList<List<Interface>>();

all paths currently being explored

List<Integer> explorecost = new ArrayList<Integer>();

cost of paths in explore

List<Set<String>> exploreriskgroups = new ArrayList<Set<String>>();

riskgroups of paths in explore

List<Boolean> exploreswitch = new ArrayList<Boolean>();

true if path has common link with primary path (shortestPath). In this case ExploreDevice looks for paths without riskgroups of primary path

Set<List<Set<String>>> exploredsets = new HashSet<List<Set<String>>>();

sets in this collection have a non-overlapping solution or have no solution

List<Device> done = new ArrayList<Device>();

Device which have been discoverd

List<List<Integer>> nodecost = new ArrayList<List<Integer>>();

cost to reach node with same index in done

Set<Interface> negativeWeight = new HashSet<Interface>();

Set of interface which have a negative weight according to Bhandari's algorithm.

int HOP_WEIGHT;

hop = x WEIGHT

int LOAD_WEIGHT;

1% load = x WEIGHT

int SHARED_DEVICE_WEIGHT;

weight of primary and backup path for sharing a device

int SHARED_RISKGROUP_WEIGHT;

weight of primary and backup path for sharing a riskgroup int SHARED_LINK_WEIGHT;

weight of primary and backup path for sharing a link

int CostFunction(List<Interface> path, List<Interface> primarypath, boolean costriskgroup, boolean noifcweight)

Calculates the cost of interface/link.

int TotalCost(List<List<Interface>> path, boolean costriskgroup, boolean noifcweight)

Calculates the Total cost of the path “path”. If costriskgroup is true then the weights of common riskgroups are also included. If “noifcweight” is true then interfaces have no weight in the calculation.

List<Integer> Cost(List<List<Interface>> list, boolean costriskgroup, boolean noifcweight)

Same as above, but for a single path.

void AddInterface(int path, Interface ifc, boolean costriskgroup, List<Interface> freeifcs)

Adds interface “ifc” to path no. “path”.

void RemovePathExplore(int index)

Removes path no. “index”.

void ExploreDevice(Interface from, Interface to, int path, Set<String> without, boolean costriskgroup, boolean firstpath, List<Interface> freeifcs)

Iteration loop of the shortest path algorithm.

int Smallest()

Returns path no. with the lowest current cost.

void FlagLinks()

Flag links of the first path so these get a negative weight according Bhandari's algorithm.

void RemoveCommonLinks(List<Interface> path1, List<Interface> path2)

Removes links used in both directions according to Bhandari's algorithm.

void RemoveExploredsets(Set<List<Set<String>>> todo)

Delete all sets which completely contain set current (including current)

String Ranking()

Outputs HTML code of the ranking of active lightpaths.

String Hotspots() {

Outputs HTML code of the hotspots in the network.

void AddPath(List<List<Interface>> path, int sts1capacity) {

Adds path to the network.

void AddPathParallel(List<List<Interface>> path, int sts1capacity) {

Not used

void RemovePath(List<List<Interface>> path, int sts1capacity) {

Remove a path from the network.

void SetNetwork(NetworkDescription netw, int hopWeight, loadWeight, int sharedDeviceWeight, int sharedRiskgroupWeight, sharedLinkWeight)

Init the network.

int CountInterface(List<Interface> path, String str) {

Count the number of interfaces in the path which contain str in the name.

Interface InterfaceMatch(List<Interface> path, String str, int xthmatch) {

Returns the xthmatch interface which contains str in its name.

List<List<Interface>> CalculatePath(Interface source, Interface destination, boolean protectedpath, int sts1capacity, boolean noifcweight)

Calculates the optimal path with or without SNCP.

List<List<Interface>> SRLGtree(List<Interface> from, List<Interface> to, int sts1capacity, boolean noifcweight)

The SRLG-tree algorithm.

List<Interface> Dijkstra(Interface from, Interface to, Set<String> without, boolean costriskgroup, boolean firstpath)

Dijkstra's algorithm.

List<Interface> DijkstraNotVia(Interface from, Interface to, String notvia)

Dijkstra's algorithm, excluding a device.

void PrintExplore()

Prints debug information.

boolean StringInPath(String str, List<Interface> path) {

Returns true if there is an interface with a name that contains str in the path.

boolean DeviceInPath(Device dev, List<Interface> path) {

True if dev is in path.

boolean InterfaceInPath(Interface ifc, List<Interface> path)

True if ifc is in path.

Set<String> RiskgroupsInPath(List<Interface> path)

Returns a set of riskgroups that is contained in path.

boolean DeviceOverlap(List<Interface> path1, List<Interface> path2)

True if path1 and path2 share a device.

boolean InterfaceOverlap(List<Interface> path1, List<Interface> path2)

True if path1 and path2 share a interface.

boolean RiskgroupsOverlap(List<Interface> path1, List<Interface> path2)

True if path1 and path2 share a riskgroup.

int RiskgroupsOverlapCount(Set<String> group1, Set<String> group2)

Counts the number of overlapping riskgroups.

Set<String> SharedRiskgroups(List<List<Interface>> paths)

Returns the shared riskgroups.

Set<String> SharedRiskgroups(Set<String> group1, Set<String> group2)

Returns the shared riskgroups of group1 and group2.

boolean RiskgroupsOverlap(Set<String> group1, Set<String> group2)

True if group1 and group2 have an overlapping riskgroup.

int FreeCapacityPath(List<Interface> path)

Calculates the maximum free capacity of the path.