

SYMBIONODE DATA CARRIER IN DELAY AND DISRUPTION TOLERANT NETWORKING (DTN)

Sašo Vrbinc, Boštjan Grašič, Marija Zlata Božnar, Primož Mlakar

MEIS storitve za okolje d.o.o.

Mali Vrh pri Šmarju 78

SI-1293 Šmarje - Sap

Tel: +386 1 3663226; fax: +386 1 3663227

e-mail: saso.vrbinc@meis.si

ABSTRACT

The work of N4C project involves a development of a future Internet in the areas without telecommunication infrastructure. Delay and disruption tolerant networking (DTN) was designed to replace the “legacy” Internet in the areas, where data mules are used as data carriers instead of a classic wiring. Data mules are typically computers with wireless communication device, but in this paper we describe different and very cost effective approach of data transfer over a simple storage device, such as USB key or memory card, in a DTN.

1 INTRODUCTION

DTN is a network solution for the area where no continuous network connection exists mostly due to the extreme environmental difficulties in which wired infrastructure is ineffective, or GSM and satellite network would be to expensive [1],[2],[3].

The N4C research group continues with the development of DTN2 bundle protocol, which is a standard implementation of a DTN architecture components primarily developed by the Universities [4],[5]. The indispensable part of the architecture comprises a routing protocol for best possible data transfer. PROPHET is a Probabilistic Routing Protocol using a history of encounters and transitivity [6]. In research presented in this paper a Prophet implementation developed in Luleå University of Technology has been used that is not based on standard bundle protocol [7].

Every DTN network consists of a set of static nodes and dynamic nodes (data mules) [8],[9],[10]. A typical data mule has connectivity with other nodes over Wireless LAN (WLAN) or other type of communication, such as Bluetooth. In contrast to a typical data mule, SymbioNode (described in this paper) is a storage device, and the data exchange is possible only when it's physically connected to another node. Therefore, some manual interaction is required for successful data transfer. All other tasks are autonomous and completely automatic with help of the Prophet functionality. To enable such functionality, many

problems arise, especially with automatic device detection and prophet running environment.

1.1 Basic concept

When a storage device is attached to the node, the system needs to assure appropriate running environment for new prophet instance. Every prophet instance requires one network interface for communication with other nodes. Since the storage device has no networking capabilities, a new or free existing network interface on the node is required.

Figure 1 describes the concept of a solution, where two prophet instances see each other in the network and possibly exchange data. From Prophet #1 perspective, another node appeared in the network and communication can be established just like with the “real” node.

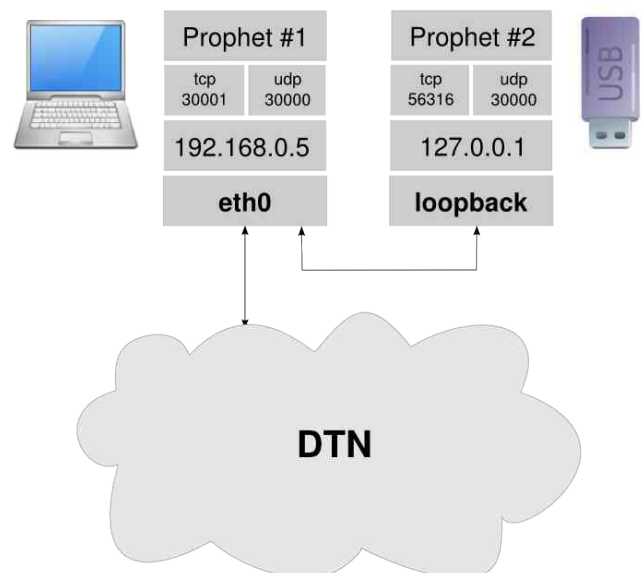


Figure 1: *SymbioNode* concept

SymbioNode represents Prophet #2 instance, which occupies the loopback device. It can communicate with Prophet #1, or with any other DTN node in the network.

Many different solutions have been tried to create a virtual network interface, including Virtual LAN(VLAN), but the only workable solution was loopback interface that successfully established communication with other nodes in the network. The given solution supports only one attached SymbioNode at a time since only one loopback device exists on the system. To avoid such limitation, the virtual IP addresses can be used, but its use in our test environment the bundle transfer occasionally failed for unknown reason.

1.2 Automatization requirements

The operating system must provide a system for auto detection of newly attached or detached storage devices. Most OS, provide such system, which are in principal more or less similar, however, in contrast to Windows Autorun, Linux has more convenient and configurable Hotplug system [11].

Each node should include a special “Hotplug” script that ensures appropriate environment for running SymbioNode Prophet #2 instance, that later makes an exchange of data between the local node, or any other in the network.

The following summary of requirements must be provided by the OS and the script for complete automatic data transfer:

- auto detect an USB key when added or removed
- mount the device to appropriate location
- provide network virtual interface or loopback device
- start prophet, and watchdog for maintenance processes

1.3 Pros and Cons

The SymbioNode technology has some great advantages over standard DTN node. Generally, SymbioNode gives the following advantages and disadvantages.

Advantages:

- very low cost system
- especially suitable for “production” of several mule nodes for test bed
- complete DTN functionality
- quick installation & configuration
- very suitable for any kind of “long range” transportation, for example, using postal services for storage device delivery between the nodes or deliver storage device between tipi and village router
- current script implementation has only been tested with Prophet, however it should be feasible with DTN2 as well
- support for different OS on the same storage device

Disadvantage:

- Physical contact is needed through USB interface which can be a difficulty if routers are placed outdoor (due to precipitation, condensation, ...)

2 AUTOMATIC USB KEY DATA TRANSFER

The target platform was chosen based on the requirements, and OS functionality. Linux provides a convenient and very configurable system for running services on device plug/unplug. Moreover, it also offers a quick and powerful scripting language Bash, which can be used for implementation of all necessary requirements summarized in the previous section. Therefore, support for automatic data transfer can be enabled with appropriate Linux system configuration and custom Prophet Script.

2.2 Prophet script configuration

The script was primarily written for “Hotplug” system, which is running on openWRT [12]. Moreover, it's fully compatible with “udev” [13] and no customization is required. Configuration of the prophet startup script is done over variables at the beginning of the script. Full description of the prophet script variables is in Table 1.

SCRIPT_LOCATION SCRIPT_PATH	Location directory of the startup script. Full path of the startup script.
DELAY	The script automatically searches for mounted device and since the mount is not instantaneous, the variable defines waiting period for the device become available by the kernel.
VIP_ENABLED	If you want to use Virtual IP then set this flag to 1
LOG_PATH	This log holds information about running prophet invoked by the SymbioNode. Each entry has the following information: <prophet_proc_id> <network_if> <device_name_path> <watchdog_proc_id>
DEVICE_DIR	System device directory. Standard location is /dev/
SEARCH_DEV	Regular expression for searching the attached device.
PROPHET_DIR PROPHET_BIN PROPHET_CONF	When device is mounted at location defined in fstab, the prophet binary is executed with command: <mount_point>\$PROPHET_DIR\$PROPHET_BIN. IP1 defined in \$PROPHET_CONF is required if \$VIP_ENABLED flag is set to 1.
PHYSICAL_IF_NAME	If \$VIP_ENABLED this variable defines an interface used for Virtual IP configuration.
VLAN_IF_PREFIX	VLAN interface name can vary with a platform. Most often used syntax is \$ {PHYSICAL_IF_NAME}.<id> or just vlan<id> The <id> is found automatically.
WATCHDOG_DELAY	Watchdog live period in seconds. When time expires prophet is terminated and storage device unmounted for safe removal.
SUSPEND_CMD	This variable is part of the power management and it's used as part of the locking system, described later in this section.

Table 1: Description of Prophet script(prophet.sh) variables

The script is also relying on mount configuration of the storage device. Therefore, an entry in /etc/fstab is required to ensure consistency with the absolute paths in prophet.ini(configuration file of prophet), such as DFTPPATH or STORAGEPATH, otherwise Prophet will not function properly. Both paths are mandatory in prophet configuration, since they define the location of

incoming/outgoing traffic and storage location where received bundles are finally stored.

Standard fstab entry for USB key looks like this:

```
/dev/sdb1 /MEIS2 ext2 rw,sync 0 0
```

File: */etc/fstab*

Sometimes it's safer to address partition by ID and consequently avoid complications with standard notation. Most Linux distributions provide partition addressing by ID and it's recommended to use it. The usual location that contains links to all storage devices and its belonging partitions is in */dev/disk/by-id/*. Every device has its own unique ID and unlikely any conflicts will ever appear. The syntax in *fstab* is the same as in previous example except the device partition location and name is different.

```
/dev/disk/by-id/usb-Verbatim_STORE_N_GO_07980809E465303B-0:0-part1 /MEIS2 ext2 rw,sync 0 0
```

File: */etc/fstab*

Mounting the device to appropriate location is crucial for SymbioNode to work correctly. When standard *fstab* notation is used for addressing device name, an incorrect mount point can be assigned by the kernel if mount point already exist in the kernel. This will lead to inconsistency with prophet configuration, and result is a broken prophet.

2.2 System Autorun configuration

On different Linux distributions and platforms, the Autorun systems differ slightly. Most frequent systems are *udev* [13] and *Hotplug2* [14] that offers customization of actions, invoked by kernel events from various physical devices. Events are device specific, and in the case of USB key two types of events(attached/detached) can be generated. In general, every removable device produce the same type of events, but all event information are specific to a different device.

Autorun Prophet Script using udev

Udev functionality gives a convenient way for selective running of user space scripts when device is attached or detached. Basically, *udev* is a rule based manager and a special rule to handle attached/detached USB device is needed. The default location of rules is in *"/etc/udev/rules.d/"*, and usually already exist some rules provided by the distribution. All rule files are in form:

```
<number[0-99]>-<name>.rules
```

If more than one exists, they are handled according to sequence number[0-99]. Custom rules have usually higher numbers. Therefore, prophet rule file should look like *"88-prophet.rules"*. Rules syntax will not be described here since many good references exists on the web.

SymbioNode can be configured for various types of storage devices(USB key, USB hdd, CF, SD cards, ...), indeed if host machine supports it. We used USB keys for testing purposes and *udev* configuration should be as follows:

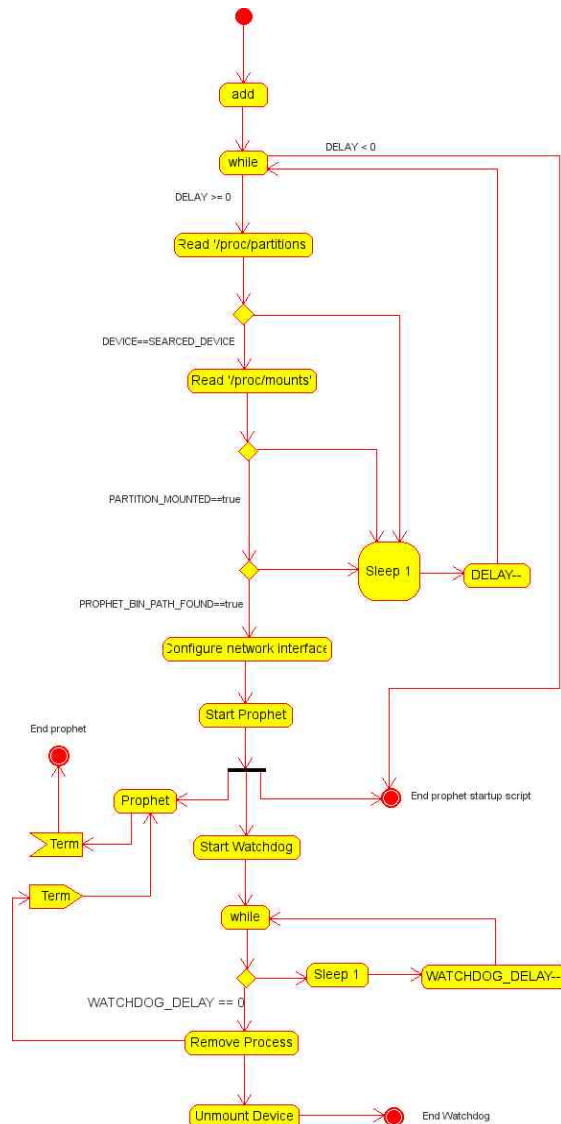
```
KERNEL=="sd[b-z][1-9]", ACTION=="add|remove", RUN +="/lib/udev/prophet.sh %k"
```

File: */etc/udev/rules.d/20-symbionode.rules*

KERNEL – Name of a device defined by the Kernel. Usually one storage device is already present as *sda* or *hda*, and the rule should be appropriately adjusted. USB storage devices are represented in the system in form *"sd[a-z][1-9]"*. The rule above applies for all added storage devices on SCSI interface except for the existing 'sda' device which is usually the system disk, and its partitions.

ACTION – With this condition it's possible to catch only certain events from kernel, in our case attach or detach of a certain device.

RUN – Path to the application or script that executes operations for the attached device. The operator **%k** is



evaluated by the kernel, and stands as a name of a device, which is passed to the script as an argument.

Figure 2: *Prophet script activity diagram*

Autorun Prophet script with Hotplug2

The "Hotplug2" system is predecessor of udev, and therefore slightly more rigid. It can still be found on some Linux distributions, mostly intended for embedded devices.

Primarily, "Hotplug2" doesn't have a strong rule based system, and therefore target device needs to be found with custom program, usually a bash script. System only separates Kernel events based on device subsystem type. Each subtype has a belonging directory in /etc/hotplug.d/, such as usb, that contains scripts for handling newly added/removed device. Nevertheless, script name paradigm and calling routines are same as in "udev".

In contrast to udev, the Prophet script has one additional task to perform when Hotplug2 is used. It is necessary to find attached storage device in order to run prophet successfully. Figure 2 shows activity diagram that depicts whole procedure of the prophet script, when a new storage device is added. If storage device was found, the script configures network interface(if virtual IP is selected), executes prophet binary, and starts the Watchdog. The watchdog ensures that Prophet process terminates, and unmounts the storage device after certain time period expires. Afterwards, the device is ready for safe removal. If device was removed before expiration time, watchdog attempts to remove the prophet process, and cleans the mount points, usually with a 'lazy' unmount. Sometimes device partitions remains in the kernel even after 'lazy' unmount. This will prevent to mount the device to appropriate location on next insertion. To avoid such flaw, it's recommended to change mount points in fstab configuration to appropriate device unique ID, indeed if it's supported by the system.

3 CONCLUSION

We developed script application and appropriate system configuration which makes automatic data transfer possible using Prophet with a storage device as data carrier. We used such system in our testbed μ -GaRaMo (Portable Gamma Radiation Monitor) [10],[15],[16] and the results are very promising. The system of data transfer showed a great reliability without any loss or corrupted data. Even when the system was not used correctly, it remained stable without damaging the filesystem of the SymbioNode nor the native system.

The system features are suitable for DTN region, where a large amount of data can be transferred over great distances with very low expenses. In the near future the system will also be upgraded with support of DTN2 reference implementation.

4 ACKNOWLEDGEMENTS

The work was done under the contract:

References

- [1] V. Cerf, R. Kahn. A Protocol for Packet Network Intercommunication, IEEE Trans. Communications, vol. 22, no. 5, pp. 637–648, May 1974
- [2] S. Farrell, V. Cahill. Delay- and Disruption-Tolerant Networking, Artech House, ISBN: 1-59693-063-2, 2006
- [3] K. Fall, S. Farrell. DTN: An Architectural Retrospective, IEEE J. Selected Areas in Communications, vol. 26, no. 5, pp. 828–836, 2008
- [4] K. Scott, S. Burleigh. Bundle Protocol Specification, IETF RFC 5050, www.rfc-editor.org/rfc/rfc5050.txt, Nov. 2007
- [5] Delay-Tolerant Networking Research Group. DTN2 code, <http://www.dtnrg.org/wiki/Code>, Jul. 2010
- [6] A. Lindgren, A. Doria, E. Davies, S. Grasic. Probabilistic Routing Protocol for Intermittently Connected Networks, <http://tools.ietf.org/html/draft-irtf-dtnrg-prophet-06>, Jul. 2010
- [7] Luleå University of Technology. PRoPHET Routing Protocol home page, <http://prophet.grasic.net/>, Jul. 2010
- [8] E. Davies, A. Doria. D2.2: Functional Specification for DTN Infrastructure Software, Folly Consulting and Luleå University of Technology, N4C project, <http://www.n4c.eu/Download/n4c-wp2-023-dtn-infrastructure-fs-12.pdf>, Jul. 2010
- [9] A. Doria, M. Udén, D. Pandey. Providing connectivity to the Saami nomadic community", Proceedings of the 2nd International Conference on Open Collaborative, Design for Sustainable Innovation (dyd 02), Bangalore, India , December 2002
- [10] E. Davies, D2.1 N4C System Architecture, Folly Consulting, N4C project <http://www.n4c.eu/Download/n4c-wp2-004-sys-arch-04.pdf>, Apr. 2009
- [11] Linux Hotplug Project. Linux Hotplugging, <http://linux-hotplug.sourceforge.net/>, Jul. 2010
- [12] OpenWrt Wireless Freedom. Homepage, <http://openwrt.org/>, Jul. 2010
- [13] G. Kroah-Hartman. Kernel Korner-udev & mdash Persistent Device Naming in User Space, Linux Journal, No. 122, Jun. 2004
- [14] Hotplug2 project, <http://isteve.bofh.cz/~isteve/hotplug2/>, Jul. 2010
- [15] M. Udén, S. Grasic, J. Näslund, M. Z. Božnar, B. Grašič, S. Vrbinc. D8.3 Test bed creation Methodological report, Luleå University of Technology, N4C project, http://www.n4c.eu/Download/n4c-ltu-40-D8_3-1_0.pdf, Feb. 2010

[16]M. Z. Božnar, A. Doria, M. Udén. D8.1 PART A: Preparation and Planning of Tests PART B: Technical Kick-Off Plan PART C: Performance / research indicators for project review, MEIS d.o.o., Luleå Institute of Technology, N4C project, http://www.n4c.eu/Download/D811_1_Test_plan_w_tech_meet_n4c-wp8-1-2_6.pdf, Sep. 2008