Selection and Publication of Network Interface Cards in Multihomed Pervasive Computing Devices

Alberto Cortés-Martín*, Carlos Garcia-Rubio*, Celeste Campo*,
Estrella M. García-Lozano* and Alicia Rodriguez-Carrion*
**Department of Telematic Engineering*
University Carlos III of Madrid
Madrid, Spain
Email: {alcortes, cgr, celeste, emglozan, arcarrio}@it.uc3m.es

***Abstract*—Many modern devices come with several, heterogeneous, network interface cards (NICs). However, simple operations like transferring data flows to the cheapest NIC or to one with enough Quality of Service (QoS) are awkward tasks on most Operating Systems. In this paper, we discuss the criteria to select the proper NIC for a given data flow. We also present a new Operating System service, called *netqos*, to publish data and figures of merit for these criteria. The main objective of *netqos* is providing relevant information to applications and middleware about NIC selection criteria, isolating them from the idiosyncrasies of the many QoS gathering tools and allowing to choose the proper NIC to fit their needs. We have built this new service as a synthetic file system for the Linux kernel. We describe our experiences in using it in a real-world scenario and the practical and inherent limitations of this approach.**

I. Introduction

You just get home from work, and your wearable computer tries to switch on the lights of your hall. Your device is still using the GPRS NIC that it was using while you drove there. However, it would be preferable to use your home Wi-Fi instead: that way your connection to your hall lights will circumvent your residential gateway firewall, save some battery and may even be cheaper.

This simple example illustrates the main topic behind our research: how do you select the proper NIC to use in a device with several NICs?

Of course, straightforward solutions can solve some of the problems easily: in the proposed scenario, a device that just switches to your home Wi-Fi, whenever it is available, will solve the problem. Actually, many modern smart phones do this automatically. However, simple, hard-wired solutions are not enough in scenarios that are more complicated:

As you approach your hall, your wearable computer detects your home Wi-Fi connections and switches to it. The device turns on the lights of your hall just fine, but the download of your favorite sitcom is delayed, as your son is saturating your home ADSL connection with his P2P activity. Also, the upload of tomorrow's travel plan to your car is halted, as your car can only be reached through Bluetooth while it is shut off.

Generic scenarios in pervasive environments require NIC selection strategies based on diverse criteria, and the user should not be bothered in the process. The main topics of this paper are what NIC selection criteria are relevant and how to expose them to the Operating System (OS), middleware, and applications for easy automation of the selection process.

Multihomed devices are devices with several network interface cards (NICs). In the past, a network router was the classical example of a multihomed device. Nowadays, all end-user mobile devices are usually multihomed; having several, heterogeneous NICs allows an always-connected approach to networking and opens the door for service continuity through vertical handovers.

Operating Systems have a long tradition in supporting several NICs, and provide convenient abstractions to isolate applications from the number and nature of the available hardware interfaces. However, these traditional infrastructures lack the dynamism, context-awareness and self-healing capabilities required to support pervasive computing: the responsibility of choosing what NIC to use for all communications is laid upon the user, and on most OSs, some expertise is required.

Using more advanced schemes to benefit from the plethora of NICs in a device is almost impossible even for advanced users. How to choose the best NIC for each network flow, given your location and battery status? The common approach to NIC selection in mainstream OSs is too restrictive and force pervasive middleware and applications to fight against the OS multihoming support instead of taking advantage of it.

In pervasive computing environments, it is not feasible to burden the user with these decisions [1]: automation is needed. This means we need a comprehensive list of NIC selection criteria and some OS API to access them programmatically; in this paper, we present both.

Section II of this document describes the abstractions, infrastructure, and criteria used by mainstream OSs to decide what NIC to use in multihomed devices, and emphasize their limitations for pervasive computing. Section III of this document list and discuss several new NIC selection criteria

that should be taken into account on pervasive computing environments, along with the difficulties we met trying to consider them. Section IV presents a new Operating System service to publish NIC selection criteria for the benefit of middleware and applications. We explain design decisions, along with its practical and underlying limitations in pervasive environments.

II. The common approach to NIC selection

The most common approach to NIC selection is the routing table. The routing table is a kernel data structure that holds what destinations are reachable through each NIC, among other things. There are three types of routing table entries according to how many hosts they stand for:

- **Host routes** or point-to-point routes; tell which NIC to use for single, unique, destinations.
- **Subnetwork routes**; associate all destinations in a subnetwork to a certain NIC.
- **Default routes** are used when all other entries failed to match a destination address.

To keep the routing table small, a few subnetwork routes, or even a default route, is preferred over many host routes, whenever the network topology allows it.

If more than one NIC fulfill all Layer 3 requirements to reach certain destination, most Operating Systems use simple solutions like the following ones:

- Use the route with the smallest *metric* (i.e. number of hops to destination). This is the old UNIX and current Windows approach (Windows 2000 and successors [2]). The metric concept only makes sense for point-to-point routes, as each destination in a subnetwork route may have a different number of hops; therefore, the concept of metric is becoming a catchall placeholder that reflects the degree of preference the host administrator has when many routes to destination are possible. Some Linux routing daemons still use this metric concept to reflect what they know about the network topology ([3]).
- Use the most specific entry to the destination, that is, choose host routes over subnetwork routes, and just use default routes as a last resort. This is the current practice in modern UNIX/Linux systems ([3], [4]).
- Use the first route that comes up in the table, ignoring the rest. This is what Windows NT does ([5]).

Any of these three approaches are quite naive for multihomed devices in pervasive scenarios. We would like to consider additional criteria for NIC selection, like price, QoS, security or user preferences.

A. Common alternatives to routing tables

Routing tables lack the necessary expressive power to cover all these new criteria, some of them are not even Layer 3 concepts. Proposed alternatives, like the ones in [6]–[10], share a common workflow pattern:

1) Measure and compile relevant data and traces, like the bandwidth and loss rates of a Wi-Fi interface.
2) Process and calculate figures of merit with enough expressive and comparison power, like quality of experience on a video streaming.
3) Handle external events to allow for some kind of context awareness, like knowing when the device is disconnected from the power plug and starts using the battery.
4) Interpret some kind of *policy description language* that will drive the final decision
5) Use an expert system to apply the policies from step 4 to the data gathered in steps 1, 2 and 3.

Operating Systems services are especially apt for storing data and traces from step 1 and the notification of the events from step 3.

However, mainstream OSs lack services to store data and traces from step 1, therefore middleware has to build such services from scratch. Applications and measure programs are thus coupled and dependant on this middleware. A modular approach will speed up development, and make applications more portable, avoiding ad-hoc solutions in userspace.

III. Survey of new NIC selection criteria

In multihomed devices, more than one NIC can fulfill the Layer 3 requirements to reach a destination. Currently, the OS routing table resolves this conflict with simple policies ignoring several important factors:

- **Link Layer information**: Like signal to noise ratio, maximum bandwidth or battery consumption. As an example, it will be interesting to discard the NICs that cannot offer the minimum bandwidth required by an application, or discard NICs with a high power consumption while in battery mode.
  We recommend a certain degree of isolation from the particular details of each link Layer; processing these data should not require a deep understanding of each NIC low-level details to allow an easily comparison of these values between different NICs.
  Many of these data will depend on how the device is used, for example, battery consumption of wireless NICs depends on the send-receive ratio or the presence of other devices nearby. Many of these details, as they are difficult to predict, may be replaced by standard figures from common scenarios.
- **Additional information from the IP Layer**. Other IP Layer elements can be taken into account to classify a communication; For example, the encapsulated transport protocol, differentiated services or the traffic class and flow label from IPv6. For example, route all data from VoIP applications through certain NIC.
- **Information about the session and application Layer**. For example, route all HTTP traffic through a

public network and all SSH traffic through your office corporate network. Getting this kind of information about the session and application Layer is not always possible, due to encryption or the use of non-standard ports. Applications can help by issuing explicit notifications about their data flows.

- **Information about the service provider**: cost, AAA policies, port filtering, firewalls, usage policies or contractual bandwidth; As for example, route all podcast downloads through a cheap service provider. [6] describes some security issues about how service providers can publish their usage restriction guidelines.
- **Security**. Some physical and link Layers are more insecure than others are, or they provide too much information about the sender. Using cryptography, protocol obfuscation and steganography in Layer 3 protocols and above, may not be enough; the simple fact of revealing your presence, approximate location or the statistical pattern of your traffic can be a threat on certain high security scenarios. There are also political considerations about routing your data through certain countries or organizations. [11] includes a brief description of these problems in military scenarios.
- **Link reliability** is an important consideration for some applications, as civil emergency alert systems or police force communications. Even if multihomed devices could handle path failures seamlessly, the downsides of a handover can discourage the use of non-reliable links.
- **Content distribution networks**. Paid subscriptions to content distribution networks may be only accessed through certain service providers. Applications using these services should only use interfaces from where the service can be reached.
- **QoS** criteria for applications have a strong end-to-end component, so they can be hard to collect. If the bottleneck is not in the access network, end-to-end QoS parameters will not be related to QoS on the first hop.
- **User and application preferences**. User and application preferences are generally expressed as policies, as they often combine several of the already mentioned criteria. On multiuser devices, similar data flows may be routed differently due to different user policies. OS administrators may impose restrictions on how NICs are used through these same user policies. Very high-level policies should be easy to issue, for example, "use the cheapest interface for all present and future communications".

It is quite complex to extend the functionality of the current routing tables to make them aware of all these criteria and policies. Routing tables simply do not have the required expressive power. The description of user or application policies is a complex topic by itself, and the decision mechanisms must understand the language used to express the policy rules. [12] and [13] describe two languages for the definition of routing policies of data flows in multihomed devices.

```
; tree /sys/kernel/netqos
.
|-- figures/
|   |-- bw/
|   |   |-- eth0
|   |   |-- eth1
|   |   |-- ...
|   |   `-- units
|   |-- loss/
|   |   |-- eth0
|   |   |-- eth1
|   |   |-- ...
|   |   `-- units
|   `-- ...
|-- policies/
|   |-- on_battery/
|   |   `-- preferred
|   `-- ...
`-- version
```

Figure 1. Partial file listing of the *netqos* file system.

IV. A FILE SYSTEM TO PUBLISH NIC SELECTION CRITERIA

We believe that publishing criteria for NIC selection must be an OS service. From the point of view of applications, there are two immediate ways to take advantage of such a service:

- Applications can choose which NIC is best for their needs, and request the OS to route their traffic through it. This is the main target of the service.
- Applications can adapt to current NIC capabilities if no better NIC is found.

We propose a synthetic file system[1] to store and publish NIC selection criteria.

The contents of the files in this new file system are data and figures of merit that middleware and applications can read or write. We organize these files in a hierarchical structure using regular directories. The name of these files and directories are straightforward representations of the kind of data they store and to which NIC they belong. We have named this new OS service *netqos*[2].

Once mounted, the *netqos* file system looks like in Figure 1. The figure shows a (partial) file listing form our test system, which has several NICs (`eth0`, `eth1`...). From now on, we will refer to the files and directories form our

[1] see section IV-D

[2] The name is misleading, but we still use it for historical reasons

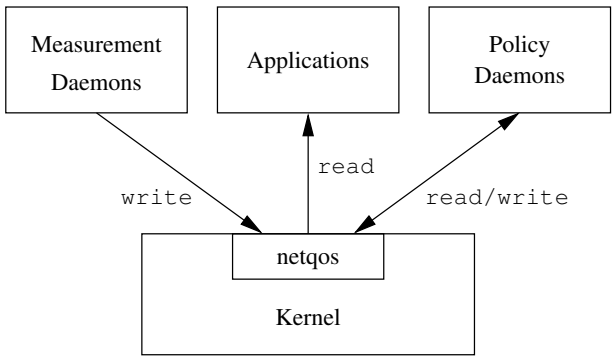

Figure 2. Processes that use *netqos* and the operations they typically perform.

synthetic file system by its relative path to its mounting point.

The file `version` holds the file system version number; processes using our file system must check this file to avoid compatibility issues between versions.

Directories like `figures/bw/` or `figures/delay/`, store files hosting the corresponding information for each available NIC. Each of these files is named after their corresponding NIC.

The files `figures/*/units` hold the measurement unit that the rest of the files in the given directory are using.

The information shared by *netqos* is quite raw: a string representation of the last value of several NIC selection criteria for each enabled NIC. Complex decisions and strategies need derived figures: for example, systems with history or hysteresis thresholds. They can be implemented by external policy modules that feed from *netqos* and fetch their own digested figures to applications through their `policy/*/preferred` file.

A. *What processes use* netqos

Figure 2 shows what type of processes might use *netqos*.

- **Measurement daemons** that gather data about NIC selection criteria figures. They will write their measurements to the files served by *netqos*. For example, a daemon measuring the delay of the network interface `eth0` once by second will write values onto `figures/delay/eth0` each second.
- **Applications** that use NIC selection criteria. They will read the files served by *netqos*. For example, a video stream server could adapt the quality of the video being served to the bandwidth available on the NIC it is using or may ask the kernel to route its packets through a more capable NIC.
- **Policy daemons** monitor several figures, and calculate complex figures of merit to identify the preferred NIC. They publish their results through its `policy/*/preferred` file. For example a policy daemon that recommends the cheapest NIC will read all `figures/price/*` files and write onto `policy/cheapest/preferred`.

B. *An example of use*

Data read from the *netqos* files will be a user readable string representation of the internal data values and will include an ending `'\n'` for better user experience.

Data written to *netqos* does not need to end in `'\n'`, but it will do no harm either. A user can read or write files from *netqos* using any of the commands available to access the file system.

```
; cd /sys/kernel/netqos/
; cat figures/loss/eth0
0.003
; cat figures/loss/units
% of packets loss
; echo 2.34 > figures/loss/eth0
; cat figures/loss/eth0
2.34
```

Of course the file system can also be read and written programmatically through the standard file system API of your OS (`open(2)`, `read(2)`, `write(2)`, `close(2)`).

C. *Why a file system?*

There are several communication interfaces between the kernel and userspace processes that can be extended to support this new service: new system calls, new ioctl variations, asynchronous notification using signals, netlink or ordinary sockets and synthetic file systems.

We have chosen to use a synthetic file system because its developer API is simple and universally well know (just read and write files). This makes application integration extremely easy while keeping the service portable to any OS with file system support.

From a user point of view, there is also a long tradition of well-known applications that can handle the file API: for example, `ls(1)`, `cat(1)`, or shell IO redirections. This allows easy scripting of the service and quick prototyping of new applications.

D. *Implementation*

We have built *netqos* as a module for the Linux kernel 2.6.29, using `sysfs` ([14]) support for synthetic file systems. The full source code of *netqos* is available at https://github.com/alcortes-uc3m/netqos.

Sysfs is a virtual file system that exports Linux kernel information about devices and drivers to userspace. These in-memory files may be accessed with the same system calls or utility programs as regular files and directories on disk. It is the equivalent of `procfs` ([15]) for devices and drivers.

A synthetic file system is the generic denomination for a hierarchical interface to non-file objects and information that appear as if they were ordinary files. `Sysfs` is particularly apt for building synthetic file systems.

The core of the implementation is finished and working, although the current figures of merit and organization of the hierarchical representation is intentionally open and flexible. We are interested in receiving feedback to include whatever figures may be interesting, and to adapt the hierarchical representation to make it simple and useful.

The current NIC selection criteria used by *netqos* has been chosen to fulfill the requirements of the CELTIC "Easy Wireless 2" international research project[3]. The main scenario of the project involves the continuity of service of a video streaming to a user terminal while it is roaming between heterogeneous wireless technologies using vanilla video streaming applications ([16]).

Netqos will support asynchronous polling through the standard `poll(2)` and `select(2)` system calls. Processes will be able to *subscribe* to the files they are interested in and receive asynchronous notifications about changes in their data. This allows applications to react easily to changes in the capabilities of the NIC they are using or to request the OS to change their flows to other NICs.

E. Limitations

Dynamic interfaces. *Netqos* does not support dynamic interfaces as it only builds files and directories for the NICs present when the module is loaded. When a new NIC is enabled or an old one is disabled, *netqos* will not notice it. We will solve this fundamental limitation in future releases by making the module aware of kernel notifications about modifications in NICs status. A high-level `events` file could be useful to notify applications about hierarchical modifications in the file tree.

Userspace policy modules. NIC selection policies must be keep out of the kernel. The current implementation of *netqos* only provides a single publication point for userspace policy modules, through a high-level `preferred` file. *Netqos* will be extended to support dynamic inclusion of userspace policies.

Multiple Network Namespaces ([17]) are not supported. The file system only considers the default network namespace. Multiple Network Namespaces support is still recent and not commonly used. We have not address this problem yet, but a good solution may be to have different *netqos* for each network namespace using the new `sysfs` support for network namespaces.

End-to-end NIC selection criteria. *Netqos* is not useful for publishing end-to-end NIC selection criteria: for example, the available bandwidth on the path to a particular video streaming server.

The collection of end-to-end NIC selection criteria is more complex than collecting internal data about the device, and sometimes needs the collaboration of the corresponding endpoint. On top of that, a service for publishing end-to-end figures must be

1) Highly dynamic. As communication endpoints comes and goes quickly in the life of a device.
2) Extremely scalable. The number of endpoints can grow insanely high.

We think that a file system is not the right tool for such requirements.

[3]http://www.celtic-initiative.org/Projects/EW-2/default.asp

V. Conclusions

The work presented in this paper is based on our experience working with multihomed devices. We have described a comprehensive list of NIC selection criteria and the lessons we have learned by struggling to use some of them for mobility support in real-world scenarios with vanilla applications in mainstream OSs.

We have developed a new OS service, *netqos*, to overcome current OS limitations and to improve pervasive applications support and creation. This new service exposes some NIC selection criteria through the well-known file system interface.

Netqos proved to be useful and facilitated the development and integration of third-party applications and middleware.

However, *netqos* is far from complete; it only supports the most relevant NIC selection criteria in our current research scenarios. We are openly looking for contributions and recommendations to cover more generic scenarios.

We are also currently working to add support for dynamic interfaces and asynchronous notification to applications, as they are fundamental features for widespread acceptance.

We plan to improve *netqos* support for the dynamic inclusion of policy daemons, allowing third-party modules to implement their own policies in userspace.

Our most pressing concern is *netqos* limitations to expose end-to-end figures. The file system interface seems to lack the required dynamism and scalability to present end-to-end figures. We plan to conduct further research on this topic.

Acknowledgments

Our research has been made possible thanks to the CELTIC "Easy Wireless II" project. This work was partially founded by UC3M and DGUI in the framework of the project CCG10-UC3M/TIC-4992, and by the Spanish Ministry of Science and Innovation within the framework of the project TEC2010-20572-C02-01 "CONSEQUENCE". Our thanks and appreciation to our assigned shepherd for his patience, and valuable guidance.

References

[1] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[2] M. Tulloch, *Windows 2000 Administration in a Nutshell*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, California, USA, CA 95472: O'Reilly & Associates, Inc., 2001.

[3] C. Hunt, *TCP/IP Network Administration (3rd Edition; O'Reilly Networking)*. O'Reilly Media, Inc., 04 2002.

[4] M. A. Brown. Guide to ip layer network administration with linux. [Online]. Available: http://linux-ip.net/

[5] Microsoft Inc., Ed., *Microsoft Windows NT Resource Kit*. Redmond, Washington, USA, 98052-6399: Microsoft Press, 1995, vol. 2.

[6] J. Ylitalo, T. Jokikyyny, A. J. Tuominen, and J. Laine, "Dynamic network interface selection in multihomed mobile hosts," in *Hawaii Intl. Conf. on System Sciences (HICSS'03)*, vol. 9. Washington, DC, USA: IEEE Computer Society, 2003, p. 315.

[7] A. Peddemors, H. Eertink, and I. Niemegeers, "Communication context for adaptive mobile applications," in *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 173–177.

[8] P. Pawar, B. van Beijnum, A. Peddemors, and A. van Halteren, "Context-aware middleware support for the nomadic mobile services on multi-homed handheld mobile devices," in *Proceedings of the 12th IEEE Symposium on Computers and Communications, ISCC 2007*. IEEE Computer Society, July 2007, pp. 341–348. [Online]. Available: http://doc.utwente.nl/64494/

[9] K. Wac, "Towards qos-awareness of context-aware mobile applications and services," in *On the Move to Meaningful Internet Systems 2005: OTM Workshops*, ser. Lecture Notes in Computer Science, Meersman, Robert and Tari, Zahir and Herrero, Pilar, Ed. Springer Berlin / Heidelberg, 2005, vol. 3762, pp. 751–760.

[10] J.-Z. Sun, J. Sauvola, and J. Riekki, "Application of Connectivity Information for Context Interpretation and Derivation," in *Proceedings of the 8th International Conf. on Telecom, ConTEL'05*, vol. 1, June 2005, pp. 303 – 310. [Online]. Available: http://dx.doi.org/10.1109/CONTEL.2005.185880

[11] J. Palet, M. Diaz, C. Olvera, A. Vives, E. Fleischman, and D. Lanciani, "Analysis of IPv6 Multihoming Scenarios," Internet Engineering Task Force, Internet-Draft draft-palet-multi6-scenarios-00.txt, Jul. 2004, work in progress. [Online]. Available: http://tools.ietf.org/id/draft-palet-multi6-scenarios-00.txt

[12] K. Mitsuya, K. Tasaka, R. Wakikawa, and R. Kuntz, "A Policy Data Set for Flow Distribution," Internet Engineering Task Force, Internet-Draft draft-mitsuya-monami6-flow-distribution-policy-04.txt, Aug. 2007, work in progress. [Online]. Available: http://tools.ietf.org/id/draft-mitsuya-monami6-flow-distribution-policy-04.txt

[13] C. Larsson, M. Eriksson, K. Mitsuya, K. Tasaka, and R. Kuntz, "Flow Distribution Rule Language for Multi-Access Nodes," Internet Engineering Task Force, Internet-Draft draft-larsson-mext-flow-distribution-rules-01, Jul. 2008, work in progress. [Online]. Available: http://www.ietf.org/internet-drafts/draft-larsson-mext-flow-distribution-rules-01.txt

[14] P. Mochel, "The sysfs filesystem," in *Proceedings of the Annual Linux Sumposium*, vol. 1, Jul 2005, pp. 203 – 207. [Online]. Available: http://www.linuxsymposium.org/archives/OLS/Reprints-2005/mochel-Reprint.pdf

[15] T. J. Killian, "Processes as files," in *Proceedings of the USENIX Summer 84 Conference*. USENIX Association, 1984.

[16] VLC: Open-Source Multimedia Framework, Player and Server. [Online]. Available: http://www.videolan.org/vlc/

[17] J. Corbet. Network namespaces. [Online]. Available: http://lwn.net/Articles/219794/