DNIS: A Middleware for Dynamic Multiple Network Interfaces Scheduling

Ahmed Saeed, Karim Habak, Mahmoud Fouad and Moustafa Youssef
{ahmed.saeed, karim.habak, mahmoud.fouad, mayoussef}@nileu.edu.eg
Nile University, Smart Village, Egypt

1. Problem and Motivation

Many of today's devices are equipped with multiple network interfaces that could be used to connect to the Internet (e.g. WiFi, Bluetooth and 3G). Those interfaces vary in many ways, including bandwidth, availability and cost. Selecting only one of those interfaces is not always the best solution to fulfill the needs of different applications deployed on current devices. For example, a VoIP application will require larger bandwidth while browsing should be cost effective. Current operating systems (e.g. Windows and Linux) selects only one of the available interfaces to be used for all the device's traffic, which results in a clear underutilization of the device's resources. Previous solutions suggested for bandwidth aggregation of the different interfaces usually include changes in the TCP/IP stack, which are exposed to the application developer, at either of the communicating parties and/or introducing devices in between, which cannot be widely adopted easily. Our proposed solution, DNIS, is a software solution that helps a device utilize all its interfaces by distributing the connections different applications make to the Internet on the available interfaces in a transparent way to both the user and the application developer. DNIS also provides an interface for allowing the user to optionally provide strategies and preferences as hints.

2. Background and Related Work

Different approaches have been proposed to achieve bandwidth aggregation. We classify those approaches based on the protocol stack layer modified to achieve aggregation. Application layer approaches require each application designer to include the logic of detecting and utilizing the different interfaces in his/her application. This gives the application designer the flexibility of defining the exact utilization function his/her application needs. However, this approach is not suitable for legacy applications and it puts a lot of burden on the application designer.

Transport layer and network layer approaches require either changes in legacy servers to be able to provide reliability over connections established through different interfaces [1], or require a service provider that acts as a proxy for the device using multiple hosts and legacy servers [2][3][4]. Link layer approaches (e.g. IEEE 802.1AX-2008) basically require homogeneous links, which is not available in mobile environments. Alteration of sockets implementation was suggested by [5] which made sockets aware of the available interfaces, and allowed sockets to distribute its traffic on available interfaces according to some utility function. However, this approach requires changes in the packets' header and changes in legacy servers to have a matching sockets implementation.

DNIS provides a middleware that does not require changes to the current applications nor effort on the application developers side. Moreover, it does not require any special devices or changes to the legacy servers.

3. Approach and Uniqueness

DNIS approach is based on a per TCP-Connection scheduling. This provides an out-of-the-box personalized solution to bandwidth aggregation that doesn't require any modification to legacy application or servers. In order to achieve its goals, DNIS needs to keep track of applications' behavior, user preferences, and interfaces capabilities to make meaningful assignment of TCP connections to certain interfaces. We start by describing the parameter estimation process, followed by the scheduling algorithms, and finally discuss our implementation.

3.1 Parameters Estimation:

For estimating the parameters of the network interfaces, DNIS keeps track of the current utilization of the interface, error rate, maximum bandwidth, and buffer size. Other metrics, e.g. [6], are also being investigated.

For the applications, DNIS keeps track of the average number of bytes sent and received per unit time, maximum number of bytes sent and received per unit time, and the application type (realtime (e.g. Skype), browser (e.g. Firefox), unclassified). The application type can be estimated based on the executable name of the process, the ports it uses, and/or its traffic pattern.

In addition, DNIS provides a GUI to allow the user to manually configure the system to provides preferences and/or strategies that can help in estimating both the interfaces and applications parameters.

3.2. Scheduling Algorithms

3.2.1. Maximum Throughput Scheduler

For a new connection, the maximum throughput scheduler assigns it to the network interface that will maximize the system throughput. This is equivalent to assigning the new connection to the interface that minimizes the time needed to finish the current system load in addition to the load introduced by this new connection. This algorithm depends on two variables that are computed by the estimation module:

1. Expected time for a certain interface to finish its current load = $\frac{\sum_{i=1}^{n} load_i}{bandwidth}$ where n is the number of connections using this interface, $load_i$ is the estimated remaining load for connection "*i*" on this interface, and "*bandwidth*" is the estimated bandwidth of this interface.
2. Expected traffic that the new connection will produce, based on the history of the application creating this connection.

Figure 1 shows an example of the operation of the maximum throughput scheduler. At time t=0, a new connection (P1) is requested. Hence there is no load currently in the system, the interface with the highest bandwidth (IF1) is selected as the one that will finish the system load first. At time t= 2, a new connection request arrives (P2) with an estimated load of 180 KB. Interface 2 is selected to be used by this connection, even though it has a lower estimated bandwidth, as it will lead to finishing the overall system load earlier. The process is repeated at time t=4 and t=7.

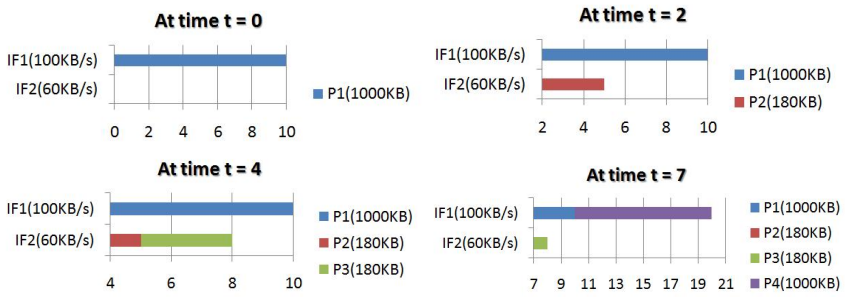

Figure 1

3.2.2. Baseline Algorithms

We also implement two simple scheduling algorithms, the Round Robin and Random schedulers, as a baseline for comparison with the Maximum Throughput scheduler. The Round Robin scheduler assigns connections in a turn to the different connections, regardless of their load or the interface condition. The Random scheduler assigns connections randomly to interfaces.

3.3. Implementation

We implemented DNIS in the Windows Vista OS. DNIS implementation is based on a Windows networking API feature called Service Provider Interface (SPI). SPI defines two different types of service providers: (1) a Transport Service Provider and (2) a Name Space Service Provider. DNIS is implemented as a Transport Service Provider. Data transfer protocols are implemented through a chain that has different layers. The base layer, called the "Base Protocol", e.g. TCP, is responsible for how the protocol works. Other layers, like our service, called "Layered Protocols" are responsible for handling high level traffic control as shown in Figure 2.

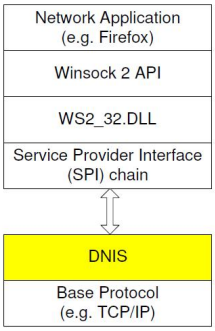

Figure 2

We implemented our DNIS middleware on the Windows Vista OS as a Layered Service Provider (LSP) [7] which is installed in the TCP/IP protocol chain in the Windows operating system as shown in Figure 1. LSP is a service which is responsible for intercepting and processing socket-based actions. That way we can intercept connections created by any application and bind those connections to the most suitable interface. This also allows DNIS to gather data about both applications and interfaces.

3.4 User Interface

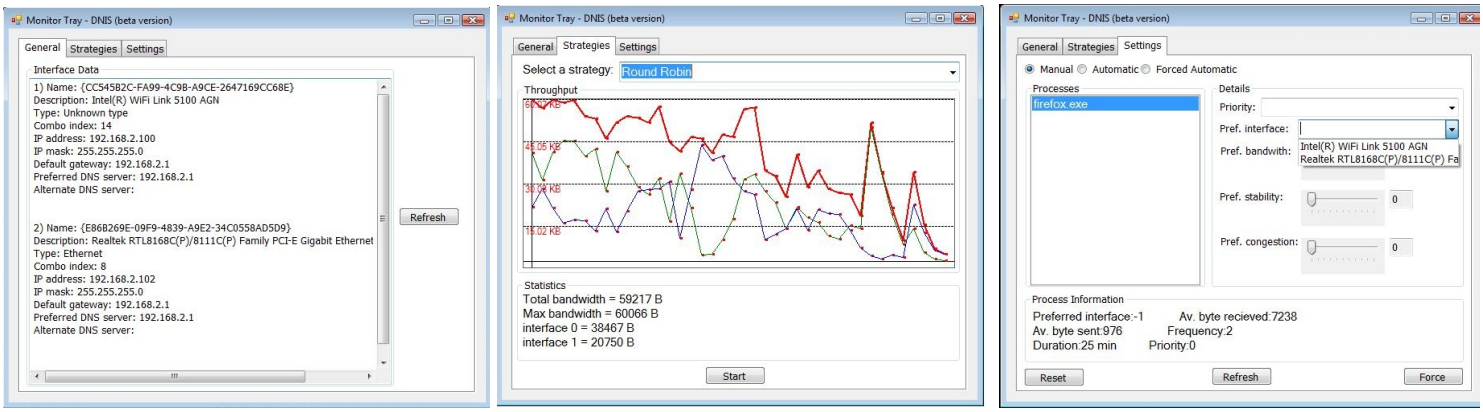

Figure 3

To be able to collect user preferences and help the user to track the performance of the system, DNIS provides a simple user interface (Figure 3). Using that interface, the user can also choose the scheduling algorithm, preview available interfaces and force his own parameters to the system.

4. Results

To evaluate DNIS, we used two laptops: (1) A Fujitsu Siemens laptop (AMILO Pro V3405) running Windows XP as a server.(2) An HP laptop (HDX 16) running DNIS on Windows Vista with a custom client for generating traffic. Both machines were connected through a wireless router (SMC7904WBRA2) such that the server connected to the router with a 100Mbps Ethernet link and the client configured to connect to the router with an 11Mbps wireless link and a 10Mbps Ethernet link to simulate client side bottlenecks, which is usually the case. Each experiment was run for 8 minutes and repeated a number of times to generate the data points. The client establishes new connections with the server according to three Poisson processes representing three classes of applications. The first class represents long-lived connections with a file download size of 150 MByte (rate= λ_{long}), the second class represents medium lived connections with a file download size of 5 MByte (rate= λ_{medium}), while the third class represents short lived connections with a file download size of 250 KByte (rate= λ_{short}).

We compare five scedulers: using only Wi-Fi, using only Ethernet, the Round-Robin scheduler, the Random scheduler, and the Maximum Throughput scheduler. Figure 4 shows the average goodput attained on different loads. As expected, utilizing more than one interface always attains higher goodput than using one interface. The Maximum throughput scheduler, which is based on user behavior tracking and interfaces characteristics, attains a higher goodput than algorithms that do not take those heuristics into account. When the system load is high, the gap between the different multiple-interface schedulers decreases as the system becomes saturated. The advantage of the maximum throughput scheduler is evident with low to medium loads. The figure shows that the maximum throughput scheduler can perform up to 54% better than using only a single interface.

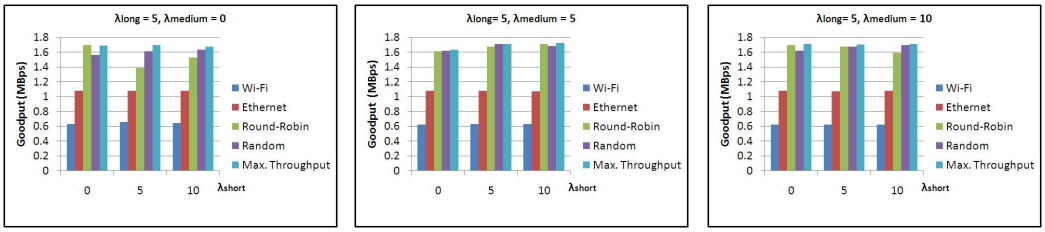

Figure 4
λ_{short}: the average number of instances of small load programs
λ_{medium}: the average number of instances of medium load programs
λ_{long}: the average number of instances of big load programs

5. Acknowledgment
This work was supported in part by a grant from the Egyptian National Telecom Regulatory Authority (NTRA).

References
[1] L. Magalhaes and R. Kravets, *Transport level mechanisms for bandwidth aggregation on mobile hosts*, in: Proc. IEEE ICNP'01 (Riverside, Nov 2001).
[2] R. Chebrolu and A. R*ao, A network layer approach to enable TCP over multiple interfaces*, Wireless Networks Volume 11, Issue 5; Pages 637 - 650 (September 2005)
[3] H. Hsieh and R. Sivakumar, *A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts*, in: Proc. ACM MOBICOM'02 (Atlanta, Sep. 2002).
[4] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt and S. Banerjee, *MAR: a commuter router infrastructure for the mobile Internet*, in: Proc. of the 2nd international conference on Mobile systems, applications, and services, Pages: 217 - 230 (2004)
[5] H. Sakakibara, M. Saito and H. Tokuda, *Design and implementation of a socket-level bandwidth aggregation mechanism for wireless networks*, in Proc. of the 2nd annual international workshop on Wireless internet, WICON'06.
[6] M. Allman, W. Eddy, and S. Ostermann, *Estimating loss rates with TCP*, SIGMETRICS Perform. Eval. Rev., Vol. 31, No. 3; Pages12 - 24 (2003).
[7] W. Hua, J. Ohlund, and B. Butterklee, *Unraveling the Mysteries of Writing a Winsock 2 Layered Service Provider*, Microsoft Systems Journal, No. 61; Pages 96 - 113(1999).