

# On Porting `iperf` to Windows Mobile and Adding BlueTooth Support

Alex Kogan  
Department of Computer Science  
Technion, Israel  
sakogan@cs.technion.ac.il

## Abstract

This paper presents high-level details of two contributions to `iperf`, a modern tool for testing network performance. The first contribution is a port of the tool to the Windows Mobile operating system, while the second one is the extension of the tool to support BlueTooth communication. The paper also includes results of experiments carried out by the modified tool in various configurations of real networks created by laptops and mobile phones.

## 1 Introduction

`iperf` [5] is a modern tool for testing network performance, widely adopted by industry and academia (e.g., [1, 3, 6, 7]). It carries performance measurements by creating TCP and/or UDP data streams and measures the throughput of the underlying network. It is an open-source project written in C and C++ and was developed originally by the National Laboratory for Applied Network Research (NLNR).

Dealing with OS-specific networking APIs, the latest official `iperf` sources<sup>1</sup> can be compiled for Unix-like (i.e., Unix flavors, including Linux) and Windows operating systems. This paper reports on the work done to port the `iperf` tool to the Windows Mobile operating system (OS) and extend the tool to support the BlueTooth communication stack. The extended tool can be found at <http://www.cs.technion.ac.il/~sakogan/iperf>. The capabilities of the original and extended tool are summarized in Table 1. The paper also reports on the evaluation of network performance carried

---

<sup>1</sup>At the time of writing, the latest official version is 2.0.4.

| OS             | TCP/UDP                  | BlueTooth |
|----------------|--------------------------|-----------|
| Unix-like      | <input type="checkbox"/> | N/A       |
| Windows        | <input type="checkbox"/> | ■         |
| Windows Mobile | ■                        | ■         |

Table 1: The capabilities of the original and extended tool.  states the feature supported in both the original and extended tool, ■ states the feature supported only in the extended tool.

with the extended tool on mobile phones running Windows Mobile and on devices equipped with a BlueTooth radio.

## 2 Technical details

### 2.1 Windows Mobile porting

The port consists of two separate applications:

**iperfWM** – the original **iperf** tool compiled for Windows Mobile .Net Compact Framework. The source was modified to redirect all the output (standard error and output streams) to files instead of a console, and to use appropriate APIs available in .Net Compact Framework instead of those available exclusively in .Net Framework.

**runnerWM** – a GUI tool written in C#, which actually runs the **iperfWM** tool and displays the produced output.

The user interacts directly only with the **runnerWM** application. Its screenshots are shown in Figure 1.

The **runnerWM** application allows a user to choose the executable file of the **iperfWM** tool, specify any standard command line parameters and, finally, run the tool. **runnerWM** starts the **iperfWM** tool as a separate process and spawns a thread that periodically checks whether the output files produced by **iperfWM** have been changed; if yes, the thread displays the updated log in the main application window. In order to avoid frequent context switching between the **iperfWM** and **runnerWM** tools, the log refresh period was set to 1 second. This log can also be saved for further investigation. In addition, the run of the **iperfWM** tool can be stopped at any moment from the **runnerWM** application (this feature is particularly useful when running **iperfWM** in the *server* mode).

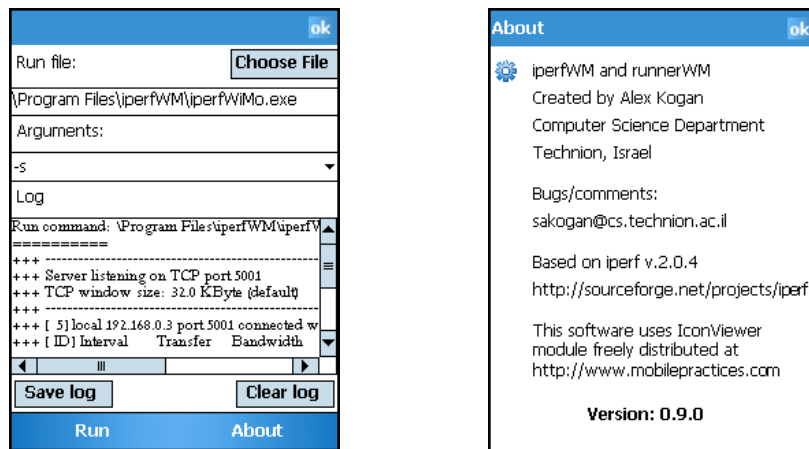


Figure 1: Screenshots of the `runnerWM` application. Due to the lack of support for multi-colored lines in UI text boxes, the log lines extracted from standard output stream are preceded by "+++" signs, while the lines extracted from standard error stream are preceded by "!!!" signs.

It should be noted that, although written specifically for the purpose of `iperf`'s porting, the `runnerWM` application can be easily generalized to run any console executable and display its logs in a convenient way.

### Command line interface

The ported version supports the same set of options recognized by the original `iperf` tool, with one exception: since the output of the tool is already redirected into a file to be read and displayed by the `runnerWM` application, the user is not allowed to specify the `-o` option, which instructs `iperf` to redirect its output into a file.

## 2.2 BlueTooth support

The `iperf` tool was extended to support Microsoft BlueTooth communication stack provided natively by Microsoft starting from the introduction of Service Pack 2 to Windows XP. The choice of this particular stack was mainly motivated by the need to enable `iperf` on devices running Windows Mobile OS, which features native support for this stack as well. This choice limits the portability of the provided extension, limiting its usage to Windows XP or higher and Windows Mobile 6 or higher. In addition, Microsoft's stack supports only one BlueTooth transport protocol, namely

Radio Frequency Communications (RFCOMM), out of at least four protocols available in other stacks [4]. Nevertheless, adding support for another Bluetooth stack, e.g., a popular Broadcom Bluetooth stack [2], should be similar and a relatively simple technical task. Additionally, RFCOMM being a robust general-purpose and reliable protocol, is the primary choice for any application requiring Bluetooth communication [4].

The Bluetooth API provided by Microsoft extends the Windows Sockets 2 API, used by `iperf` for TCP/UDP programming. In particular, it includes all standard operations required for creating sockets, establishing a connection, sending/receiving data and closing the socket. The main differences are, essentially, at the syntactic details (e.g., the address family being `AF_INET` and the protocol being `RFCOMM`) and at the layout of basic data structures, such as a socket address structure (for more technical details, refer to the excellent book on Bluetooth programming by Huang and Rudolph [4]). As a result, adding Bluetooth support required intervention in a very limited number of places in the original source, fitting the parameters in the calls for socket creations and casting to/from appropriate data structures to handle information on sockets, such as network address and port number.

Bluetooth support is controlled by a special `#define HAVE_BLUETOOTH` directive, which is currently enabled for Windows and Windows Mobile platforms.

### Command line interface

In order to specify that Bluetooth communication is required, a special command line flag `-z` was introduced. This flag should be used in conjunction with other standard flags, such as `-c` for the *client* mode and `-s` for the *server* mode.

An `iperf` user has several options to specify the server for Bluetooth communication when running the tool in the *client* mode. First, there is the usual option of giving the full address of the corresponding server as a parameter for `-c` option. In the case of Bluetooth, this is an 48-bits address, which should be given in the `XX:XX:XX:XX:XX:XX` format. Sometimes, however, the address might not be available or require non-trivial operations to be discovered. For this purpose, the `iperf` tool can be invoked to perform the *Bluetooth server discovery* by specifying zero for the server address (i.e., `-c 0` option). In this case, the client will make an inquiry for any available Bluetooth devices around and display their addresses and assigned names. In case only one device is found, the client will try to connect to it; otherwise, it will finish the discovery and exit. Finally, the last option to identify

the server is to specify some portion of the server's address, e.g., the middle (or last) five digits in the format `X:XX:XX`. In this case, the client will run server discovery and will try to connect to the server with the address containing the given portion.

It is worth mentioning that the discovery process takes a few seconds to complete, and although it is not counted for network performance, it prolongs the duration of experiments. Note also that since the RFCOMM protocol has only 30 ports available, running `iperf` with Bluetooth communication will normally require to specify a port from the allowed range, instead of using the default one (5001).

To summarize, the common invocation of `iperf` in a Bluetooth *server* mode would be:

```
iperf -z -s -p 15
```

while in a Bluetooth *client* mode it would be:

```
iperf -z -c 00:09:dd:10:6e:2f -p 15
```

or:

```
iperf -z -c 0 -p 15
```

or:

```
iperf -z -c 6e:2f -p 15
```

### 3 Performance evaluation

In this section, we detail some results achieved by the modified `iperf` tool when run on Windows Mobile-enabled phones and on Windows devices equipped with Bluetooth radio.

#### 3.1 Setup

The measurements were taken in networks consisting of two devices placed roughly one meter one from another, in the same room on the same surface without obstacles in between. The networks were configured in one of three settings: (1) WiFi network configured in ad-hoc mode, where two devices communicate directly; (2) WiFi network configured in access-point (AP) mode, where two devices communicate through a wireless router; (3) Bluetooth network, where two devices communicate directly through Bluetooth

radio. All experiments were opted to avoid any interference from other WiFi or BlueTooth radios by using WiFi radio channels free of any communication and by performing the actual measurements only during night hours. In addition, only the radio under measurements was on during the experiments, and the GSM radio of the mobile phones was always switched off.

For the WiFi networks, we experimented with both UDP and TCP transport protocols. For the BlueTooth network, we experimented with (the only supported) RFCOMM protocol. In all experiments with UDP, `iperf` was configured to send packets at a rate of 60Mb/s, which exceeds the link bandwidth (54Mb/s). This is in order to achieve the maximal available link throughput (the sending rate in TCP or RFCOMM protocols cannot be controlled, and `iperf` always strives to achieve the highest possible rate).

We used two Lenovo T61 laptops and two Samsung i900 (Omnia) mobile phones. All these devices are equipped with built-in WiFi 802.11b/g and BlueTooth 2.0 radios. In the AP network configuration, we also used a Linksys WRT54GL wireless router.

The length of each experiment was 60 seconds. Each reported throughput was calculated based on the average of 10 experiments run in exactly the same configuration. All throughput numbers are in Mb/s units. We also report the minimum and maximum throughput. The standard deviation for throughput in experiments with WiFi was below 0.7Mb/s (with the majority of results having standard deviation below 0.1Mb/s). In experiments with BlueTooth, the standard deviation was below 0.07Mb/s. In our figures, *laptop-mobile* means that the report is for a network consisting of one laptop and one mobile phone, and the laptop serves as an `iperf` client (i.e., sends data) while the mobile phone serves as an `iperf` server (i.e., receives data).

## 3.2 Results

### WiFi ad-hoc network

Figures 2(a) and 2(b) present results achieved in a WiFi-based network configured to operate in ad-hoc mode with TCP and UDP, respectively. For the former, it can be seen that configurations involving mobile phones achieve only 53 – 61% of the throughput that can be achieved when two laptops communicate, while all configurations are far from the theoretical throughput that can be provided by the 802.11g standard (54Mb/s).

The UDP protocol exposes two interesting phenomena. First, its performance in any configuration was lower than that of TCP. This is opposite to

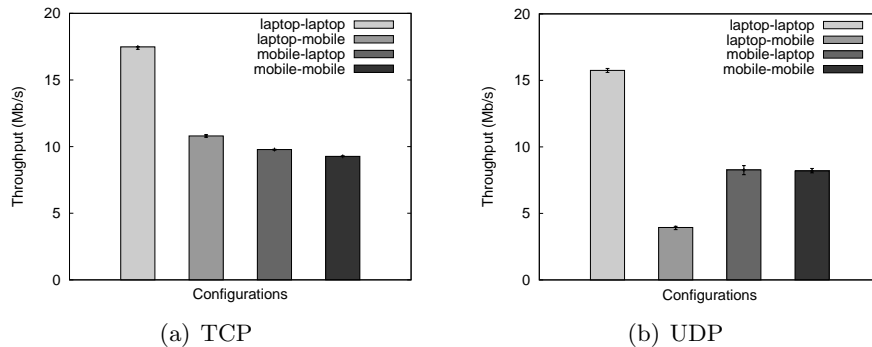


Figure 2: Throughput of the WiFi-based ad-hoc network.

the common expectation that an unreliable protocol should trade reliability for performance. This is the outcome of the limitations of APIs provided by Windows OS, and not caused by the work done in the scope of this paper. More specifically, `iperf` requires to specify the desired bandwidth for UDP communication and tries to control the rate at which UDP packets are sent by putting the sending thread into sleep for (short) periods of time. The length of these periods is carefully calculated, and, depending on the rate specified by the user, can be as short as a few microseconds. While Unix-like systems, complying with POSIX standards, have a special function for handling such a high resolution for the sleeping interval (`nanosleep`), Windows API lacks such a function and allows only intervals of whole milliseconds. This leads to sub-optimal performance of UDP protocol in `iperf` on Windows systems since the sending thread spends more time sleeping than actually required. We currently investigate how this limitation can be overtaken. We should also note that when experimenting with Linux, the throughput of UDP reported by `iperf` in the same network configuration was clearly higher than that of TCP.

A second phenomenon is exposed by UDP in the configuration of *laptop-mobile*, where the throughput is drastically lower than that of any other configuration. This can be explained by the fact that the WiFi card of the laptop can send data at a higher sustainable speed than the card of the mobile phone is capable of receiving. Thus, many transmitted packets are dropped by the phone. Figure 3(a) shows the average number of lost messages as reported by `iperf`, and confirms this explanation.

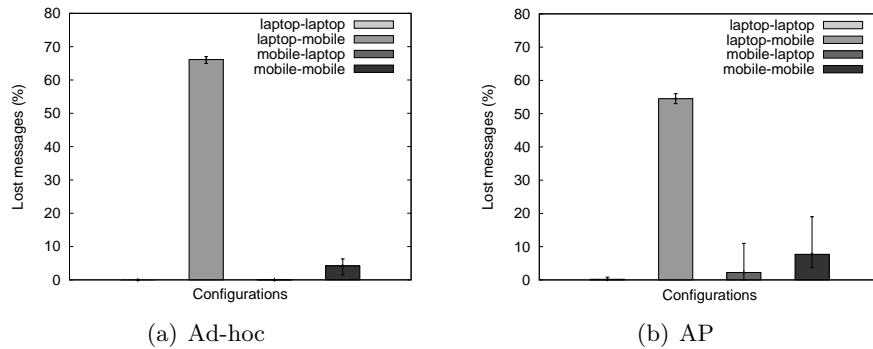


Figure 3: Percentage of sent messages that were lost by the receiver in UDP communication in WiFi-based ad-hoc (a) and AP (b) networks.

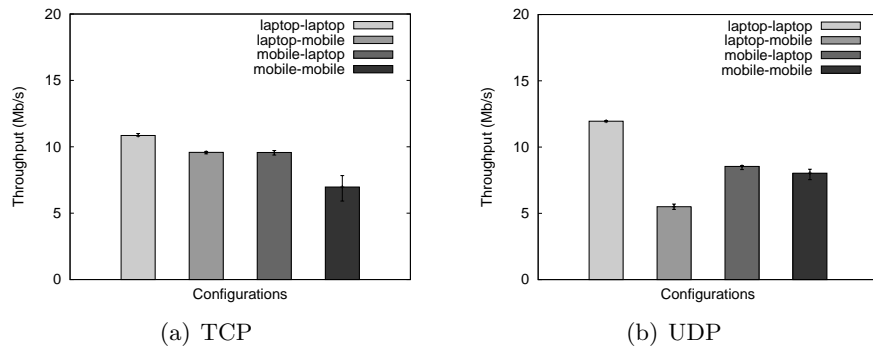


Figure 4: Throughput of the WiFi-based AP network.

### WiFi AP network

Figures 4(a) and 4(b) present results achieved in a WiFi-based network configured to operate in the access point mode with TCP and UDP, respectively. Here, the TCP performance is generally lower than in the ad-hoc case due to the fact that each packet now traverses two wireless links as opposite to one direct wireless link in the ad-hoc mode.

The results for UDP show a reduction in throughput in *laptop-laptop* configuration, which can be attributed to the presence of the additional physical wireless link. Other configurations achieve the same or even higher throughput, which can be the outcome of the imprecision of UDP measurements in Windows, as explained in the previous section. The packet loss (see Figure 3(b)) follows the pattern seen in the ad-hoc network configuration.

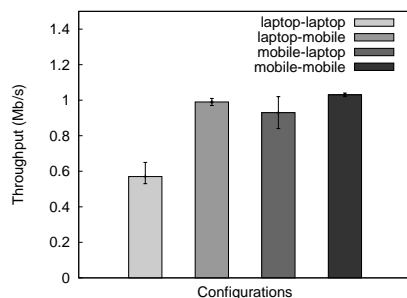


Figure 5: Throughput of the BlueTooth-based network operating with the RFCOMM transport protocol.

### BlueTooth network

Figure 5 shows the performance of a BlueTooth-based network as measured by the extended `iperf` tool. It is interesting to note that the throughput between two laptops is almost half of the throughput in any other configuration, which is completely the opposite of the results in WiFi-based networks. Understanding this phenomenon is a part of our future research.

### Acknowledgement

The author would like to thank Oran Barak for providing initial information and sources of `iperf`, including various bug fixes and extended porting to Windows. Also, the author would like to thank Roy Friedman for fruitful discussions.

### References

- [1] S. Bhandarkar, S. Jain, and A. L. N. Reddy. LTCP: improving the performance of TCP in highspeed networks. *SIGCOMM Comput. Commun. Rev.*, 36(1):41–50, 2006.
- [2] Broadcom Corporation. <http://www.broadcom.com/support/bluetooth>.
- [3] L. Cherkasova, D. Gupta, and A. Vahdat. When virtual is harder than real: Resource allocation challenges in virtual machine based IT environments. Technical report HPL-2007-25, HP Laboratories Palo Alto, 2007.

- [4] A. S. Huang and L. Rudolph. *Bluetooth Essentials for Programmers*. Cambridge University Press, 2007.
- [5] NLANR/DAST. Iperf. Available at <http://sourceforge.net/projects/iperf>.
- [6] H. Sivakumar, S. Bailey, and R. L. Grossman. Pockets: the case for application-level network striping for data intensive applications using high speed wide area networks. In *Proc. of the 2000 ACM/IEEE conference on Supercomputing*, page 37, 2000.
- [7] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman. Direct device assignment for untrusted fully-virtualized, virtual machines. IBM Research Report H-0263, IBM Research Labs, 2008.