



Mobilizing Applications: Adapting to Available Network Bandwidth

by Fred Cooper

April 15, 2004

The information contained in this document is provided for informational purposes only and represents the current view of Intel Corporation Intel on the date of publication. Intel makes no commitment to update the information contained in this document, and Intel reserves the right to make changes at any time, without notice.

DISCLAIMER. THIS DOCUMENT AND ALL INFORMATION CONTAINED HEREIN IS PROVIDED AS IS. INTEL MAKES NO REPRESENTATIONS OF ANY KIND WITH RESPECT TO PRODUCTS REFERENCED HEREIN, WHETHER SUCH PRODUCTS ARE THOSE OF INTEL OR THIRD PARTIES. INTEL EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, IMPLIED OR EXPRESS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, AND ANY WARRANTY ARISING OUT OF THE INFORMATION CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION, ANY PRODUCTS, SPECIFICATIONS, OR OTHER MATERIALS REFERENCED HEREIN. INTEL DOES NOT WARRANT THAT THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN IS FREE FROM ERRORS, OR THAT ANY PRODUCTS OR OTHER TECHNOLOGY DEVELOPED IN CONFORMANCE WITH THIS DOCUMENT WILL PERFORM IN THE INTENDED MANNER, OR WILL BE FREE FROM INFRINGEMENT OF THIRD PARTY PROPRIETARY RIGHTS, AND INTEL DISCLAIMS ALL LIABILITY THEREFOR.

INTEL DOES NOT WARRANT THAT ANY PRODUCT REFERENCED HEREIN OR ANY PRODUCT OR TECHNOLOGY DEVELOPED IN RELIANCE UPON THIS DOCUMENT, IN WHOLE OR IN PART, WILL BE SUFFICIENT, ACCURATE, RELIABLE, COMPLETE, FREE FROM DEFECTS OR SAFE FOR ITS INTENDED PURPOSE, AND HEREBY DISCLAIMS ALL LIABILITIES THEREFOR. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT OR TECHNOLOGY DOES SO AT HIS OR HER OWN RISK.

Licenses may be required. Intel and others may have patents or pending patent applications, trademarks, copyrights or other intellectual proprietary rights covering subject matter contained or described in this document. No license, express, implied, by estoppels or otherwise, to any intellectual property rights of Intel or any other party is granted herein. It is your responsibility to seek licenses for such intellectual property rights from Intel and others where appropriate.

Limited License Grant. Intel hereby grants you a limited copyright license to copy this document for your use and internal distribution only. You may not distribute this document externally, in whole or in part, to any other person or entity.

LIMITED LIABILITY. IN NO EVENT SHALL INTEL HAVE ANY LIABILITY TO YOU OR TO ANY OTHER THIRD PARTY, FOR ANY LOST PROFITS, LOST DATA, LOSS OF USE OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF THIS DOCUMENT OR RELIANCE UPON THE INFORMATION CONTAINED HEREIN, UNDER ANY CAUSE OF ACTION OR THEORY OF LIABILITY, AND IRRESPECTIVE OF WHETHER INTEL HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

Intel, the Intel logo, Pentium, Intel Xeon, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2005 Intel Corporation

Introduction

Today's mobile notebooks have battery lives of 8 hours or more, and can be connected to the Internet through a cellular GPRS connection one moment and through an 802.11 wireless access point the next. Software architects can no longer assume that the environment in which their applications run is static. The world of mobility is here and that means power, processor speeds, network availability, security, and bandwidth are all subject to change without notice. Software must be designed to best take advantage of what resources are available, when they are available. The mobile notebook is becoming the target platform for application development. This paper focuses on adapting an application's network usage to make the best use of available bandwidth. I'll demonstrate simple techniques to measure the available network throughput to determine if it is adequate for your application's changing needs. Moreover, to ensure needed data is available to the user as quickly as possible, I will share techniques to organize data transmissions with mobile network policies.

This paper discusses how to adjust an application's network usage to take advantage of the network bandwidth available. It does not cover detecting network connections, active network adapters, or adapter type or speed all of which were covered in my previous paper "[Implementing Network Detection for Mobility](#)".

Policy Management

At the heart of adapting software to its mobile environment is mobile policy management. Mobile policy management defines which data is most important, least important, which can be cached locally, which should be transmitted, when, and on which types and speeds of networks. In short, mobile policy management determines how an application reacts to its mobile environment.

This can be as simple as using an algorithm that decides to use a local database when a network connection is less than 100Kbps and a network database when the connection is 100 Kbps or greater. Such an algorithm would work well for certain types of applications without a lot of data to exchange and no security concerns. However, applications with more demanding networking needs require sophisticated policy management algorithms.

Architectural Considerations

Architecting for mobility does not lend itself well to one solution that can meet needs of all applications. There are many factors that must be considered when architecting a mobile application. For example, the more data your application needs to transfer in a short period of time, the greater the need to intelligently manage those transmissions to adjust to the available bandwidth.

In most products, it makes sense to localize the mobile policy management into a single object or component. Due to time constraints, developers often intermix the policy code (the code that decides what data to send, how much, in what order, and when) with the network transport code which is simply responsible for sending and receiving the data. Separating the policy code from the

network transport code will make it easier to increase the sophistication of the mobile policy management over time. In the initial version of an application, for example, all the mobile policies may be hard-coded into the business logic; in a subsequent release some policies may be configurable by the user. Localizing policy management in your code also makes it easier to take advantage of any system-wide mobile policy management services that may, in the future, be produced by various software vendors.

Prioritizing Data

In an application where a large amount of data needs to be shipped in a small period of time, data prioritization is essential. If the network connection is slow, it is important to get the data that is needed immediately by the user across the network first. You want to avoid your application's own data from artificially becoming a barrier to the application's ability to respond to the user. For example, in a simple email application, the first priority of data is to send the subject lines of the new messages to the email application, followed by the full text of the message the user is currently poised to read first. You don't want the application to be in a position where it has to wait for the full text of email messages, or even worse, file attachments of messages the user isn't currently viewing to be received before it can display the text of the message the user is actively attempting to view. By sending less relevant data first, followed by key data required by the application to respond to the user, an application can create a blockage of data that directly impacts the perceived performance of the application.

There are two ways to avoid this blockage: first is to properly prioritize your data. The priority of the data is a policy decision, and should be made by the policy manager. Key points to consider when setting data priority:

- **First things first.** What data is needed immediately? Evaluate your program and look at what data is really required to be transferred before the program can respond to the user. In an email program, it might be the visible text of the email the user is currently reading and the subject of other emails. In a database program it might be the data of the selected record. In a document viewer, when the user is reading the 57th page of a document, it might be just the text on that page and the number of pages in the total document.
- **Anticipate what data is needed next.** After you've met the immediate urgent need of the user, use your policy engine to decide what data should be transferred next. In an email program, maybe it's the full text of the next message after the currently open message, followed by the full text of the message that precedes the open message. In a sales person's database application, maybe it will update the other accounts owned by that sales person first, followed by accounts of other sales people. In a document reader open to page 57, page 58, 59, followed by 56.

In short, the policy manager should attempt to determine what data is most likely to be needed next by the user. Often that can be ascertained by knowing how users commonly use the product. For example, the architect of a document reader application may know that users read the next page of the document 80% of the time, the previous page 15% of the time, and scroll to a random page 5% of the time. Therefore the architect designs the policy manager to transfer two pages after the current page, followed by one page before, followed

by all other pages.

In some cases, it is necessary for the policy manager to ask the user for input on which data is most important. In these cases, a configuration dialog in which the user can choose which types of records or data is most important. For example, the user can specify the application to give a higher priority to: emails from a specific person, records from a specific account, files of a specified type, etc. Another method is to include a priority in the data itself by allowing the user to designate a record, file, email, etc as being high or low priority.

- **Getting the biggest bang for your buck, data size.** In some cases, there are trade offs to be made when prioritizing your data, and on low bandwidth connections you must decide which data will give you the biggest bang for your buck. For example, in a document reader after transferring and displaying the text for the current page with placeholders where pictures would go, the policy manager may have to make a choice, either transfer and display the images for the current page, or download the entire text of the document first. The entire text of the document may take less time to transfer than the one image, and therefore may have the larger impact on the user and be a better use of available bandwidth. If, for example, there are five records that are all of equal priority to the policy manager in all other regards, four of them are small and one is quite a bit larger, the policy manager should transfer the smaller records first which, once again, will have the biggest impact on the user.

The second technique to avoid a data blockage is to design the transport layer to be packetized and support multiple tasks. To be more specific, the transport layer while transferring data across the network for one record must be flexible enough to put a current record on hold, transfer data from a higher priority record, and then resume transferring data from the original record once the higher priority data is transferred. And this must be completed without resending any data that was previously transferred. This type of architecture allows the policy manager to better respond to the user's needs. For example, if the user views a record that is not available locally, the transfer of the record is immediately started; however, before the transfer is complete, the user switches the view to a different record that is also not available locally. Then the transfer of the first record should be put on hold, while the transfer of the actively viewed record is processed. Once the actively viewed record is transferred, the transfer of the original record may be completed.

Minimizing Data

To make the best use of the available network bandwidth, we must avoid consuming network bandwidth with unnecessary data. There are four things I would like to focus on in regard to minimizing data: (1) offline data stores; (2) differencing; (3) restarts; and (4) compression.

- **Offline data stores:** The term "offline data store" is perhaps a bit misleading, because a local copy of frequently used online data can dramatically improve the overall responsiveness of an application when it is online as well. Certainly, when a network connection is not available, or a user decides to disable the wireless radio to conserve power, having a local copy of the data is necessary to maintain an application's functionality and a user's productivity. However, quite often even when online, having a local copy of data makes it possible to only transfer changes to data across the network, hence shortening the time the user must wait for the complete data to be available. Even on a

slow network, if the differences are small, the application may remain very responsive to the user. For networks, such as cellular, that charge for access by the minute, an offline data store can actually save the user money by allowing the user to synch, disconnect, and then continue to work offline.

- **Differencing:** To put it simply, don't resend data you already have. If your application already has a copy of a record (or a file, email, image, etc) either from an offline data store or because the record was transferred earlier, don't send it again. If the record was modified, only the differences between the old and the new record need to be exchanged on the network. The entire record does not need to be resent.
- **Restarts:** Connections get interrupted and tasks get reprioritized. Whatever the cause, a data transfer may fail or be delayed, and the application needs the ability to restart the transmission avoiding resending data that has already been sent.
- **Compression:** To maximize the use of the available network bandwidth, compress data when applicable. Note that not only will compressing data make the data transmission faster, it could also save the user real money, as some types of network connections charge by the minute or Megabyte, as with certain GPRS or public hot-spots. In addition, there may be some power benefits if a wireless network card is transmitting for a shorter period of time.

Network Policy

Not all networks are created equal. Some have shared bandwidth, while others have dedicated bandwidth. Some permit a high level of security while others do not. Some cost \$4 a megabyte while others have unlimited usage. The mobile policy manager detects the type of network that is being used for the network connection and decides how, if at all, the application will make use of that network.

Below are network attributes that affect mobile policy:

- **Connection Speed:** Connection speed is perhaps the most common attribute for mobile policy managers to use. The reason is simple, it's straightforward to obtain this information (see: [Implementing Network Detection for Mobility](#)), and it's available on all adapters before a single packet is sent. Typically this number is either the speed of the modem connection, the LAN connected to the computer, or the adapter speed. It isn't necessarily the speed at which data can be transferred on the network, as the network may be in heavy use, or somewhere along the connection path is a slower router or gateway. However, it does give the mobile policy manager a good idea of the capabilities of the connection, and the types of data transmission it might want to try. For example, if the connection speed was 33.6K, the policy manager may choose not to exchange data files over 500K. If the connection speed was 56K, perhaps it would allow files up to 1Mb to be exchanged, and if the connection speed is greater than 100K all files will be updated as needed.

- **Available Bandwidth:** Connection speed is typically the maximum theoretical speed for the adapter or the connected LAN. However, available bandwidth is a computed value of how fast data actually passes through the network. If the network is overloaded, regardless of what its theoretical maximum speed is, the actual amount of bandwidth available to your application may be substantially less. Hence, a policy manager using Connection Speed alone may erroneously decide to synchronize large amounts of data on a connection that is overloaded and hence inadequate. However, if the policy manager actually monitored the available bandwidth of the connection as well, it could make better decisions.
- **Quality of Service:** Quality of Service (QOS) allows an application to request a minimum quality of its connection, peek bandwidth, rate, latency, etc. The network then reserves the resources requested along the network path. The caveat to QOS is that it requires a path in which all routers, switches, and end-nodes along the path to be QOS enabled. If this is not the case, QOS will fail. A policy manager can use this information when available to determine potential available network bandwidth. It can also decide if it can send certain types of data streams based on the quality of the connection and whether or not QOS is available on that network.
- **Security:** Keeping the user's data secure is the responsibility of every application. Each network connection may have a different level of security. For example, some data may require a VPN or an internal wired connection to send certain data, other types of data may be transmitted unencrypted over the public Internet. Some gateways and routers don't support VPNs. The mobile policy manager can decide what type of data to exchange based on the security of the current connection.
- **Type:** Knowing the type of network connection, such as wired, wireless, GPRS, modem, etc. gives the policy manager the insight that allows it to make some intelligent choices about the connection. For instance, if it is a modem connection, the policy manager can avoid doing available bandwidth tests, as the bandwidth is accurately reported by the connection speed. With the knowledge that a connection is wireless, the policy manager can expect the available bandwidth to vary frequently as the signal strength varies and other wireless connections share the access point. In knowing this, the policy manager may be more conservative on the amount of data transferred or the size of the packets.

With the right information available to your mobile policy manager, it can take that information and determine what the application will do in certain situations. Once again, there is no single solution to mobile policy management, so exactly what your application does is dependent on the data that needs to be transmitted. If, for example, a large amount of data needs to be transmitted in a short period of time, the policy manager must have a solution for low bandwidth connections. That solution should specify what data won't be transmitted in low bandwidth situations. An online database application can reduce the number of database records that are synched to only the accounts assigned to a specific user or accessed in the last 15 days on low bandwidth connections. It could exchange only text based data on low bandwidth connections and full blob data on higher bandwidth connections.

Combinations of the network attributes may factor together to make up part of your mobile network policy. For example, if the connection is wireless and a VPN is not available, don't exchange network data—use the offline data store instead.

Power Policy

Power is the final resource the mobile policy manager needs to help manage. When running on battery power, an application needs to be aware of how much power remains and what can be done to conserve it. An application can conserve power by accessing the hard disk or CD ROM less frequently, doing less computations, thus allowing the CPU to switch to a low power mode, and transmitting less data over a wireless connection, thus allowing the wireless network adapter to switch to a low power mode. The mobile policy manager in the application should make intelligent decisions to reduce power consumption when possible. It also needs to know how to respond when power is running low to ensure that local data stores are saved and network connections are terminated.

Bandwidth Detection

There are various methods for estimating the available network bandwidth of a connection. For mobility, it is important to remain focused on the level of accuracy your mobile policy manager requires to adjust the data usage of your application. To measure the available bandwidth of a network, it is necessary to inject probing traffic into the network and therefore degrade the network's performance. So the first step in detecting available network bandwidth for mobility, is determining when it is actually necessary. For example, dial-up network connections are typically limited by the speed of the modem—when the modem connects, its connection speed is reported by the interface. Hence, when the policy manager detects a modem connection it doesn't need to measure the network bandwidth.

On wired and wireless LAN networks, the bottleneck is less likely to be the network adapter, and more likely to be the available bandwidth of the network. In this case, we send probe packets across the network to determine if the available bandwidth is adequate. Once again, focus on the level of accuracy your mobile policy manager requires. For most mobile application purposes, it is not necessary to determine the maximum available bandwidth of a network, but rather if the network can meet a minimum need. The point being, that to determine the maximum available bandwidth the probe packets will have to exceed the available bandwidth of a router or switch on the path, which impacts everyone on the network. If your mobile policy manager would respond the same way if the available bandwidth were 1 Mbps or 1 Gbps, it is not necessary to consume all 1 Gbps of available bandwidth of a network to prove it is adequate. Also keep in mind the usage model of your application. If you are designing client software that will appear on hundreds of desktops in a corporate environment, IT departments would be unwilling to deploy software for which each client periodically consumed all their available network bandwidth. So if you expect multiple copies of your application to be running in the same network environment, you'll want to be conscious of how your application impacts overall network performance.

To that end, let's review a few approaches to determine if the available bandwidth is adequate for the application's needs.

Available Bandwidth using UDP

First, the basics: If I send a stream of UDP packets from one end point to the other, by simply recording the time when the first packet is received and the time when the last packet is received, I can compute throughput by dividing the bytes received by the number of seconds it took to receive them. Notice that this approach does not reveal the total available bandwidth of the network. To know if we've exceeded the available bandwidth, we look at the end-to-end delivery times. When the probe stream exceeds the available bandwidth, the network will start to queue the packets. A queued packet takes longer to reach the end point than a non-queued packet. If we send UDP packets at a rate that exceeds the available bandwidth, not only will it result in packet queuing, but the queue will grow with each successive packet. Therefore the overall end-to-end delivery time for each successive packet will increase as well. If we are not exceeding the available bandwidth, the end-to-end delivery time for each packet will be roughly the same for each packet. So to determine if we've exceeded the available bandwidth, all we need to add is some time information to the packets. The information we really need to know is when the packet was sent from the source computer and when did it arrive on the remote system in relation to the other packets in the stream. The clock starts when the first packet of the probe stream is received on the remote system. Contained in that first packet is information that tells the remote system how much time should be between packets. The remote system compares this intended arrival time of the second packet to the actual arrival time of the second packet. The difference between the two is recorded. If stream is below the available bandwidth, the difference between the intended separation of the packets and the actual received separation will remain relatively flat. However, if the probe stream exceeds the bandwidth of the network, the packets will be queued, and as more packets are queued, the difference between their intended arrival time and their actual arrival time grows. So if we have exceeded the available network bandwidth, the difference will, on average, continue to grow.

TCP Throughput

The above approach works well if you are trying to measure available network bandwidth for UDP packets; however TCP/IP is an entirely different beast. As a reliable transport protocol, TCP typically has its own buffers on both sending and receiving systems, a return channel of ACK packets, and the ability to share bandwidth with other TCP connections. For example, a network that is saturated with a single TCP connection, you would probably expect that the available bandwidth on this network is zero. However, because TCP shares the transport, $\frac{1}{2}$ of the bandwidth may be available to a new TCP connection.

Since TCP has a return channel of ACK packets, its throughput is typically dependent of the Round-Trip Time (RTT) as well as the start up time of TCP. Different implementations of TCP implementations may also dramatically affect the transfer capacity of the network.

With TCP we are unable to control the exact content of packets, the rate at which packets are sent, and if they get resent. The typical method for determining available bandwidth of a single TCP connection is to set a timer before transferring or receiving data, stop the timer when the transfer is complete and compute the throughput by dividing the number of bytes transferred by the number of seconds required to transfer them. To avoid wasting perfectly good network bandwidth, send data you wanted to transfer anyway. Once the transfer is complete, the mobile policy manager will have a better understanding of the available network bandwidth, and can determine to what extent it will continue to use the network connection.

Sample Code

Available with this paper is a sample application, **UDPNetSample**, <link to code sample page in CR 19240> which demonstrates the UDP method of measuring network bandwidth. To detect network bandwidth, the sample application must be running on both ends of the network connection. Choose a sample application on either end of the network connection and enter the IP address of the other system running the sample application, then press the Connect button. This allows the application to test that the remote address is accessible. Once connected, in the edit box at the bottom of the dialog, enter the speed in mega bits per second that you would like to test.

This sample application links to a library, **INetBWUdp**, which has a simple API for requesting network bandwidth. It is in this library, **INetBWUdp**, where the network bandwidth detection is accomplished. The simple API is defined in the **INetBWApi.h** file, where each function and its parameters are described. Here is a summary of some of the functions in this API:

- **INetBWConnect:** Called to test the IP address and establish the IP addressed used by other INetBW functions.
- **INetBWDisconnect:** Called to clear the connection state.
- **INetBWListen:** Called by the application to open a port to receive incoming BW test packets.
- **INetBWStopListening:** Close the port opened by INetBWListen.
- **INetBWTestNetSpeedAsync:** Start testing the network for a given bandwidth and return immediately without waiting for the test to complete.
- **NetBWTestNetSpeedSync:** Start testing the network for a given bandwidth and don't return until the test is complete:
- **INetBWSetNotifyWindow:** Register a window handle to receive notifications when a connection is made or a test is complete.

The source code to this library is also included, where the main class **CSocketMgr** handles all the packets exchanged as well as the actual test measurements.

The key functions in **CSocketMgr** are:

UDPTestBandwidth: This function, given a suggested bandwidth, determines the size, number and frequency of packets to send to the remote system, constructs the packets, and sends them at the determined rate.

HandleTestPacket: This function is called on the remote system each time a test packet sent by UDPTestBandwidth is received. This function simply records several key statistical

elements as the packets are received, such as: total number of packets received, total number of bytes received, time of first packet received, did this packet take longer to transmit than previous packets?

TestComplete: This function is called when the last packet of the test is received, or the test has timed out, to complete the test. This function evaluates the data accumulated during the test, computes the results, and sends them back to the source system.

Conclusion

Good mobilized applications make the best of the available network resources. For your application to best respond to the ever changing mobile world, incorporate a mobile policy manager that can decide how your software will respond to various mobile conditions. This policy manager assures that the data is transferred in the right priority order and pares it down for lower bandwidth connections. It decides when to use certain network connections and when to use a local data store. It can decide when it needs to determine the available network bandwidth and when an adapter interface speed is adequate. The algorithms described in this paper will assist you in developing code to determine the available network bandwidth and incorporating mobile policy management into your own applications.

References and Resources

[Implementing Network Detection for Mobility](#)

[Download the code samples mentioned in this paper](#)

[Windows* WM_POWERBROADCAST Messages in a Mobile Environment](#)

[Developing WinSock Applications for a Mobile Environment](#)

[Mobilized Software Developer Center](#)

