COSC 465: Computer Networking
Lab 3: BitBlaster Measurement Tool
Due: 22 February 2012, before lab

1 Overview

In this lab you will code up the sender (client) side for a UDP-based measurement tool called `bitblaster`. The receive-side code will be given to you.

The tool will send a specified number UDP packets of a given packet size and at a specified rate to a receiver. The receiver will collect the incoming packets and compute some basic statistics on the packet arrivals.

Preview: once this lab is complete, we'll be using your bitblaster tool to take measurements on paths on campus, and in the wide-area Internet (using PlanetLab). With this program, we'll be able to measure loss and delay of a packet stream emitted at a given rate.

As usual, you're welcome to work with someone else on this lab. Only one submission per group needs to be made. Please note in your source who the authors are.

2 Detailed Description

The `bitblaster` sender must accept 5 command-line arguments:

1. the receiver IP address,
2. the receiver port number (must be between 1024 and 65535, inclusive),
3. the target bitrate (r) (in bits per second, must be positive),
4. the packet size (p) (in bytes, must be between 20 bytes and 1470 bytes, inclusive),
5. and the number of packets to send (n) (must be at least 10).

You can decide exactly how the sender accepts these arguments, but the five parameters must be configurable from the command line. You should also error-check the parameters, as indicated above.

Once configured, the `bitblaster` sender will send n UDP packets to the destination address and port given on the command line. The UDP payload size of the packets will be p, and the packets will be sent such that an average bitrate of r is achieved.

Note that when a user specifies a particular packet size, the *actual* UDP packet sent on the wire will be larger. In particular, there are 46 bytes added (20 bytes for the IP header, 8 bytes for the UDP header, and 18 bytes for the Ethernet header and trailer). Thus, if a client specifies 100 bytes for the packet size, the actual packet will be 146 bytes on the wire. Your sender program should take these overheads into account when attempting to send packets to achieve a particular bitrate.

How to achieve a particular bitrate

The third command line argument to the `bitblaster` sender is a bitrate that your program should attempt to achieve when emitting packets. For example, if a user specifies 100000 bits/sec, then your program should attempt to send packets at a rate in order to achieve, on average, 100000 bits/sec.

The packets within the stream you emit should be as close to uniformly spaced as possible. For example to achieve a bitrate of 100 Kb/s when sending 100 Byte packets (1168 bit packets, when including overheads), the packets should be spaced approximately 11.7 milliseconds apart. Thus, in order to achieve a given rate, you'll need to pause between sending consecutive packets. You can use any mechanism you want to effect a delay between sending packets. One simple method is provided by the operating system via the `usleep` (or `nanosleep`) function calls. These cause the calling program to pause for a specified number of micro (or nano) seconds. Another approach is to have the program busy-wait (sit in a tight loop) for a specified amount of time. *Don't assume that usleep or nanosleep will sleep for exactly the specified number of micro or nano seconds!* Operating systems are funny creatures and you're unlikely to get exactly what you ask for. Thus, you will probably need to modulate (adjust) your sending rate slightly over the course of sending some number of packets. Relatively frequent use of `gettimeofday` to monitor your progress will probably help. (But be aware that abuse of this call can slow your program down.)

Another important issue to be aware of as you make different intermediate computations within your program (e.g., to figure out how many microseconds you need to pause between sending packets), is integer overflow. That is, you may need to make computations that would result in a larger number than can fit in a standard 32-bit integer. You can consider two options: use a `double`, or use a 64-bit integer. For the latter, you can simply use the type `uint64_t` (an unsigned 64-bit integer type).

Your program should be able to closely produce the desired rates for the following configurations:

- 10000 bits/sec (10 Kb/s) with 50 byte packets
- 100000 bits/sec (100 Kb/s) with 100 byte packets
- 1000000 bits/sec (1 Mb/s) with 100 byte packets
- 10000000 bits/sec (10 Mb/s) with 1000 byte packets
- 100000000 bits/sec (100 Mb/s) with 1470 byte packets

Use the above settings as examples for "reasonable" parameter settings. For example, achieving 100 Mb/s with small packets (e.g., 100 bytes) is not likely to be possible (packets would need to be sent at ≈ 11 microsecond intervals).

Testing your sender

To test your sender, you'll need to compile and use the `bitblastrecv` program, posted on Moodle. This program takes two command-line parameters: the UDP port number on which packets are sent, and the IP address of the sending host. You can use what ever valid port number you'd like in your testing. To obtain the IP address of the sending host, you can use the `ifconfig` program from a shell: it will show all active interfaces on a host and IP addresses assigned to them.

Initially, you should test your program on a wired network, not wireless (especially for rates above 1 Mb/s). You're unlikely to achieve high rates, and you'll be creating havoc for other wireless users. I'll only test your code on wired connections.

You'll need to come within 5% of the target rate for the sample configurations above in order to receive full credit.

Bitblaster packet and stream format

The packets that are emitted from the sender must have a particular format. Specifically, there must be 5 4-byte unsigned integers in *network byte order* at the beginning of each packet:

- `command`: The `command` value should be 0 for every packet except the last two. The last two packets should have the hexadecimal value `0xDEADBEEF`. These values will indicate to the `bitblaster` receiver that the stream is ending.
- `sequence`: Sequence numbers must be assigned to emitted packets. They may start at any value, but must be strictly increasing. It is customary (but not required) to start at 0.
- `length`: The length of the payload in bytes. (Note that this value does not include any protocol header overheads.)

- `send_sec`: The time (in seconds) just before the packet was sent. This time value should be obtained from a call to `gettimeofday` (i.e., the `tv_sec` field from `struct timeval`).
- `send_usec`: The time (in microseconds) just before the packet was sent. This value should be obtained from a call to `gettimeofday` (i.e., the `tv_usec` field from `struct timeval`).

Because of these 5 fields, the minimum packet size for `bitblaster` is 20 bytes. The remaining payload contents of the packet can be set to any value. For example, if the user requests to send 100 byte packets, the first 20 bytes will be filled in as specified above, and the remaining 80 bytes can have anything in them. The largest packet that should be sent from `bitblaster` is 1470 bytes.

Note that there is a structure defined in `bitblast.h` that is used by the `bitblaster` receiver for the packet layout. You can reuse this definition if you want in your own code.

Getting started

You're welcome to use any code you've written so far to get started (e.g., your `Blat` client code). There's an `SConstruct` file posted on Moodle that assumes your source code is named `bitblastsend.cc`, but there is otherwise no template code for this lab.

The receiver side code is posted on Moodle as `bitblastrecv.cc`. As you're testing your sender code, you can use the receiver to test whether your program is emitting packets at the desired rate (the receiver program prints out a series of statistics at the end of a run). Note that if there is excessive packet loss during a bitblaster experiment, the receiver may not receive the appropriate packets at the end of the stream (i.e., with command `0xDEADBEEF`) to know it should stop. In that case, you will have to type `^C` to stop the receiver, then try again.

3 Submission

Submit your `bitblastsend.cc` source code to Moodle. Be sure to note in the comments at the top of the file who the authors are (esp. if you work with a partner).