Chapter 8
Bandwidth-Limited Systems

Bandwidth-limited systems

- the reliability and capacity of the underlying network is not sufficient for the task at hand
- e.g.,
  - wireless communication devices
  - use of multimedia content in nw'd apps.

Outline of the chapter

- bandwidth monitoring
  - built within java.io's I/O stream classes
- general content consumer/producer model
  - having adaptive buffering for data being streamed over the network

Two flavors of limited bandwidth - 1

- application can have high bandwidth requirements (i.e., required rate of data flow is close to capacity of network)
  - e.g.,streaming high quality video for real-time playback
    - a constant, high=throughput, reliable network connection is necessary

Two flavors of limited bandwidth - 2

- Network connection has low/unreliable capacity and is insufficient for many data transactions
  - e.g., current telephone modem throughput rates are insufficient to support downloading high-quality multimedia in real time.
  - Many wireless communication devices unreliable to the point that their effective throughput << their peak throughput

Two flavors of limited bandwidth - 3

- In summary:

  Data requirement of application

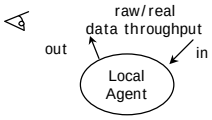
  >

  available bandwidth

Coping with Limited Bandwidth

- Monitor data throughput to detect changes in runtime environment
- Manage the bandwidth usage of the system to react to these changes

A) Monitoring bandwidth



- Raw data: is fed into/out of system at the socket or stream level
  - compressed or encoded
- Real data:

Monitoring…

- Monitoring raw data throughput
  - in order to respond to network variability (bandwidth fluctuations, loss of service, etc.)
- Monitoring real data throughput
  - in order to pick up on major fluctuations in net bandwidth usage and local resources like CPU availability while maintaining a certain performance level

How to measure performance?

- depends on application, but will typically a function of
  - responsiveness,
  - relative rate of data delivery to user,
  - etc.

Managing bandwidth

- In order to satisfy application requirements
  - A multimedia presentation with an audio track needs to ensure that the real input rate of audio samples into the local audio device is >= playback rate (in order to avoid interruptions)
  - An interactive chat client may want to balance input/output rates (so that user typing a response can see other user's response)

Managing bandwidth

- Managing bandwidth and local resources to support the type of data being processed
- Managing the nature of data itself in order to match the bandwidth and local resource profile

Example for second case

- Choosing the encoding format of the transmitted data for limited-bandwidth applications
  - tradeoff between expected bandwidth and local resource capabilities
    - choose best compression ration for low-bandwidth situations
    - choose most robust algorithm for loosy network situations

Network-level protocols to support monitoring and management in real time

- Real-Time Protocol (RTP)
  - provides a protocol layered on top of a baseline network transport layer like TCP, with header info capable of providing data timing and ordering statistics
- Real-Time Control Protocol (RTCP)
  - is meant to provide basic bandwidth management functions for RTP applications

RTP and RTCP is to be supported in Java Media Framework

Monitoring Bandwidth

- Ability to monitor effective bandwidth seen by an application
  →
- ability to adapt to variable runtime environments

Some bandwidth measures

- Average data throughput rate over a given time period
- Total data throughput over a given time period
- Estimate of time until a given amount of data will be available
- Other first- and second-order statistics on data rate and throughput over time (variances, median rate, data "acceleration")

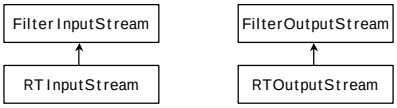
We would like to

- capture these bandwidth measures in real time
- have these measures in terms of both raw (unprocessed) data throughput and real (application) data throughput.

DataMonitor class

- provides a container for holding byte counts of data and corresponding start and stop times.
  - addSample(): for adding bandwidth measurement samples
- can be queried for statistics using
  - getAverageRate()
  - getRateFor()
  - getLastRate()

Raw Data Monitoring



- They monitor their own raw data rates using the DataMonitor class.
- After each read() and write() operation, a data point is stored in the stream's DataMonitor.

Hiesenberg Uncertainty Principle

- Measuring resources affects the measurements themselves

```
public int read() throws IOException {
   Date start = new Date();
   int b = super.read();
   monitor.addSample(1, start, new Date());
   return b;
}
```

Example: streaming an audio file from a server for local, real-time playback

- Steps: (all are in a single thread)
  - data i/o -> decoding -> writing to local audio device

Effective raw data input rate for one cycle:

$R_T = d_t / (t_r + t_d + t_w)$

+ Monitoring:

$R_T = d_t / (t_r + t_d + t_w + t_c)$

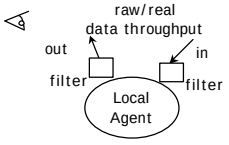
How to make t_c negligible?

- read and process large amounts of data in each cycle
  - hinders our ability to track data rate variations over time
- ignore the effect of data monitoring and read very small packets of data in each cycle
  - larger negative impact on the data rate itself

Real Data Monitoring

- Raw data monitoring does not tell us anything about whether our local data processing is keeping up with network requirements of the system
  - e.g., we may be pulling real audio data from the network fast enough, but decoding may be taking longer than expected
- Monitor the rate at which we are processing data from its format on the network to a format suitable for the local application, and vice versa.

An infrastructure for monitoring



- Filters may compress, modify, or subdivide the data passed through them
  - can be thought of as content producers/ consumers, or both.

How to construct this infrastructure?

- Develop basic interfaces for these content consumers and producers:
  - ContentConsumer: accepts data and consumes it, display data on screen, store data in a database or file, or it may feed some kind of analysis engine.
  - ContentProducer: generates data by pulling data from persistent storage, or as a product of some processing by another producer.

Chaining producers and consumers

- Each producer and consumer has a source and a destination.

ContentConsumer class

- consumeAll(): consumes data from its producer until it is exhausted
- consume(): accepts a data buffer in the form of a byte array, consumes it by calling
  - preConsume() - initialization
  - doConsume()
  - postConsume() – cleanup
- and creates a data sample for the DataMonitor associated with the consumer

Constructing Pipelines

```
ContentProducer input = new MyProducer(host,
  port);
ContentConsumer dbase1 = new
  RDBMSConsumer("jdbc:odbc://dbhost/mydata");
input.setDest(dbase1);
ContentConsumer dbase2 = . . .
dbase1.setDest(dbase2);
. . .
input.produceAll();
```

Monitoring both raw and real data rates

- Ex: Image processing pipeline

```
InputStream imgStream = . . .;
RTInputStream rtStream = new RTInputStream(imgStream);
ContentProducer source = new StreamProducer(rtStream);
```

END