Producer flow control

Producer flow control is necessary to stop clients from overwhelming a broker if messages are not being consumed fast enough, this is filed in Jira as QPID-942.

Use cases

1. Consuming Client lags Publishing Client (P2P)

Desc
This scenario is where the client consuming cannot process the data published to its queue by another client as the same rate i.e. consumption lags publication. This seems to be an almost de facto use case for P2P messaging in that by its nature (and possibly via the client code) consuming messages involves more processing than publishing i.e. send() is a less complex action than receive() or onMessage(). It can also happen when the consumer goes away.

Result
Message back up in the queues on the broker and are not being drained by the consumer. This may eventually lead to OOM as the VM cannot garbage collect the message refs. This could happen slowly over a period of time, so the MINA buffers may be empty (or at least not represent any significant amount of memory use).

2. Unconsumed messages remain in Queues (PubSub)

Desc
This is where data is being published to topics in the broker for which subscriptions exist, but no consuming client acks the messages.

Result
Messages back up in the broker and with durable subscriptions never go away. The broker OOMs are the queues are full. Again this happens over time and the MINA buffers may not be impacted. Without TTL set (and set low enough) then the data backs up in the client's sub queues.

3. Consuming Client cannot process large messages

Desc
This is where the consumer cannot process a large message sitting in its queue. This may be because it does not have enough memory or disk available for the processing, for example. It may also arise if the message is corrupted in some way i.e. malformed XML etc. The message doe not get ack'd and remains in the broker on a queue, currently surviving restart.

Result
The message(s) remain in the broker and can cause OOM, particularly when there's a burst of large messages together. An example of this is the Qlib issue where they had a spate of large messages and client side OOM precluded them being processed. It can happen slowly, and with persistent messages costs at least twice as much heap. Broker OOM follows eventually.

Plan

To implement this, the following changes are necessary:

Client
send() needs to become potentially blocking, if the producer has been flow controlled then send() should not either throw an exception (which will be the default behaviour), or it will block until the producer has been unflowed (this will only occur of a system property has been set).

Broker
When a message has been enqueued, the broker should check if the producer is publishing to a queue which has violated it's policy, if so then the producer will be flow controlled. When a consumer has had a message delivered, if the queue is no longer violating it's policy then producers will be unflow controlled. This check will occur after enqueing so as not to slow down the broker.

Queues and exchanges will have policies attached to them (queues will inherit from their exchange if they do not have one), which will specify the point at which producers should be flow controlled in terms of queue count or queue depth. These policies will be manageable over JMX, so they can be applied or removed without having to restart the broker.

The management console will also gain a "stop all producers" button to enable throttling of arbitary queues, and a "start all producers" button which will start all flowed producers.

Disadvantages of this approach

The producer will not be flowed until they publish to a queue which is violating it's policy, so if you have N producers each publishing to a queue, you will get N messages on top of the one that pushes the queue into a delinquent state.