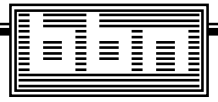
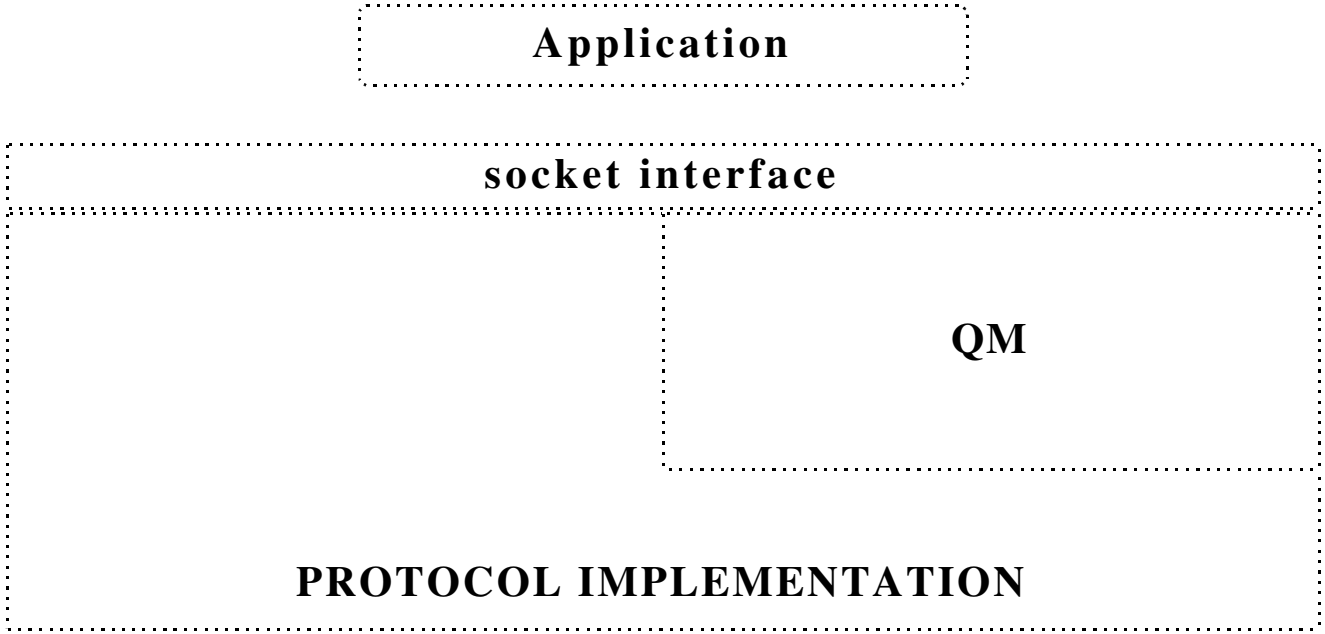
Sockets and a Quality of Service Manager

- This talk is an attempt to take the idea of a Quality of Service Manager and convert it into a concrete set of routine calls
- Much of this is probably wrong! But...
  - concrete examples help reveal problems
  - concrete examples help explain abstract ideas
- Based in part on a note I sent around in August about making sockets ready for integrated services
  - taking basic ideas from that note and extending them for a QM

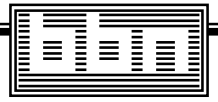


A Sketch of the Pieces



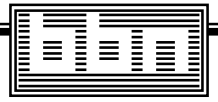
Why Put QM In Kernel?

- The QM negotiates quality of service issues
- That means it has to be integrated with the data path
  - if you change service type, you may have to adjust data handling or even the flow id
- And the data path is in the kernel



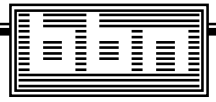
A Receiving Application

- Starts by creating a socket and binding to an address (such as the session multicast address)
- Then chooses the sender and flow id it wishes to listen to and notifies QM of the minimum QoS it desires (can be zero)
  - single flow ID per socket (just because it make state management easier)
- Then asks QM for unranked list of QoS choices for that flow and picks one
  - assuming single flow can have multiple advertised QoS



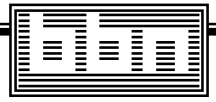
Receiving Code

```
s = socket(AF_INET,SOCK_DGRAM,0);
bind(s,(struct sockaddr *)mcast,sizeof(*mcast));
connect(s,(struct sockaddr *)src,sizeof(*src));
setsockopt(s,IP_QM,SO_FLOWID,(char *)
   flowid,&flowsize);
setsockopt(s,SOL_SOCKET,SO_QOS,(char*)
   minqos, &mqsize);
getsockopt(s,IP_QM,SO_FLOWCHOICES,(char
   *) flowlist,&listsize);
setsockopt(s,IP_QM,SO_FLOWCHOICES,(char
   *) flowchoice,&onechoice);
```



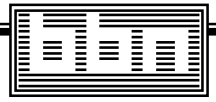
Changing the Service

- **At any time, a receiver can**
  - **get the current list of flow choices and pick a new one**
  - **change the source (by calling connect)**
  - **change the flow ID (by setting it to a new value)**
- **So if at any time the application or user decides different service is desired, application can ask for the range of choices**



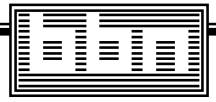
At the Source

- **The source creates a socket, may optionally bind too.**
- **Source determines basic QoS information (transmission rate, maybe delay or loss sensitivity) and general class of service (guaranteed, predictive) desired and notifies QM**
- **Source asks for flow ID and connects to multicast address**



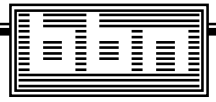
Source Code

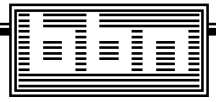
```
s = socket(AF_INET,SOCK_DGRAM,0);
qos = getqosbyname(“mpeg-2”);
setsockopt(s,SOL_SOCKET,SO_QOS,(char *)
   qos,&qoslen);
setsockopt(s,SO_QM,SO_SERVICETYPE, (char
   *)service,&servlen);
getsockopt(s,SO_QM,SO_FLOWID,(char *)
   flowid,&flowsize);
connect(s,(struct sockaddr *)mcast, sizeof
   (*mcast));
```



QM Issues

- **How wide a range of service classes can a sender advertise?**
  - **predictive, and guaranteed and best effort?**
- **How many flow IDs does this require?**
  - **if more than one, how to minimize duplicate copies of data?**
- **Can a source stop advertising a particular class of service**
  - **Presumably yes, but what does this do to receivers?**
- **How does receiver know if a QoS is better?**
  - **cost vs. bandwidth tradeoffs**
  - **library routines to compare them?**



Socket Issues

- **This interface implies one flow ID per socket**
  - **flow ID related to source address and a set of shared state**
  - **keeping one set of state per socket seemed easiest since sockets are cheap**
  - **but does mean if you're conferencing with several folks, you'll need several sockets (even if they all share the same multicast group). [But keep in mind, one socket can change sources by calling connect()]**
- **Calls to [sg]etsocketopt getting a bit complex**
  - **need new system calls?**
  - **can we keep many of them generic (such as setting QOS params)?**