CMPSCI 754 Multimedia Systems **Spring 2001**

Lecture 22: March 26

Lecturer: Prashant Shenoy *Scribe: Yan Mu*

22.1 Overload Control

Overload means that aggregate rate requirement exceeds link capacity. The key challenges are (i) how to detect overload, and (ii) decide what to drop in case of overload. If we know the semantics of the video stream, from application level, e.g. for MPEG video streams, we could drop B-frames when coming into an overload.

The assumptions are: (i) round time of each source T, T can vary, no need to be synchronized; (ii) swithes periodically detect and control overload (period=T).

22.1.1 Overload Detection

If rounds are synchronized, overload detection would be easy. If rounds are not synchronized, we have to do more jobs. We know that within one stream, rates vary in different rounds, but in one round, rate keeps the same. So we need to detect overload at the start of EACH round. See the following chart:

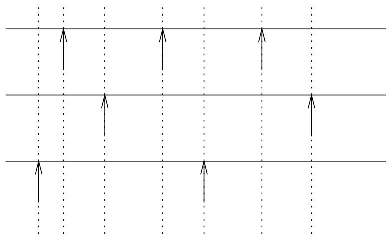

Figure 22.1: Overload Detection

22.1.2 Rate Assignment

We can calculate loss affordability, the rate that can be reduced without violating QoS as the following:

$A_i = max\{0, q_i \sum_{j=1}^{j=\gamma_i+W+1} \gamma_i^j - \sum_{j=1}^{j=\gamma_i+W+1} (\gamma_i^j - \hat{\gamma}_i^j)\}$

q_i is loss affordability, γ_i^j is the required rate, $\hat{\gamma}_i^j$ is the rate actually provided.

Rate Assignment Algorithm

The algorithm processes sub-intervals from left to right and reduces rate of sources from bottom to top. It tries to minimize the aggregate rate reduction. Its complexity is *O(N)*.

22.2 Design Issue for Integrated Services Networks

The difference between best-effort applications and real-time applications:

- Best-effort applications are elastic, does not need performance guarantees; real-time applications require guaranteed (statistically or deterministically) Quality-of-Service.
- Best-effort applications are congestion-avoidant; real-time applications are congestion-ignorant.

Internet only supports best-effort services. So there are two approaches for designing an integrated services networks:

- Change the network and application implementation, employ smarter applications over the best-effort network. This approach does not need to change the architecture of the network, the cost is relatively small.
- Change the architecture of the network. This approach will result in tremendous work and huge cost. Many people don't think it feasible.

22.2.1 Change the Implementation Approach

Use packet scheduling algorithms in routers to eliminate the interference of real-time applications with data applications. This approach is relatively easy to deploy but actually network still delivers the same class of service to all applications. There's no guarantee of QoS. Video conferencing system will not tolerate the delay of buffering. Applications will be very complicated and difficult to develop.

22.2.2 Change the Architecture Approach

This approach is easier for application development. Basically, the change of the network architecture can be transparent to the applications. But this approach requires fundamental changes in the existing network. It will be too expensive to deploy.

Why extend the service model?

The goal of network design is to meet users' satisfaction. Introduce the concept of utility = f(s,l,d,j,...), which is a function that concerns about several factors like loss rate, delay, etc.

Why we need to extend FIFO service model to Priority service model? Here's an example: Exponential server with rate $\mu = 1$; two flows with Poisson arrivals with rates $\gamma = 0.25$; $U_1 = 4 - 2d_1$ and $U_2 = 4 - d_2$. Define $V = U_1 + U_2$. System can be modeled as an M/M/1 queue. We analyze this model and get the following results:

FIFO service: V=2; Priority service: V=$\frac{8}{3} > 2$

Thus, priority service is better than FIFO service. This result can be generalized as: more class of services, the better. But anyway, the result depends on the definition of the utility functions (what's the user's preference?).

22.2.3 Need for Admission Control

Preventing overloads:

- Overprovisioning: If bandwidth becomes cheaper (cost-effective), this is simpler to do.

- Admission Control: avoid overloading by restricting admission.

Whether we need admission control depends on the utility functions:

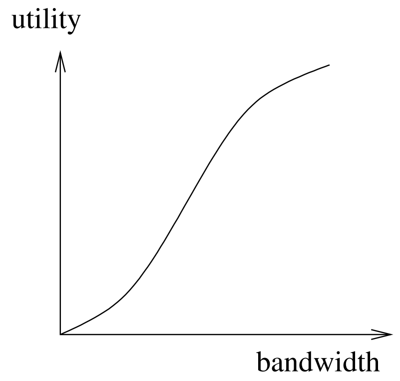

Figure 22.2: Best-effort Utility Function

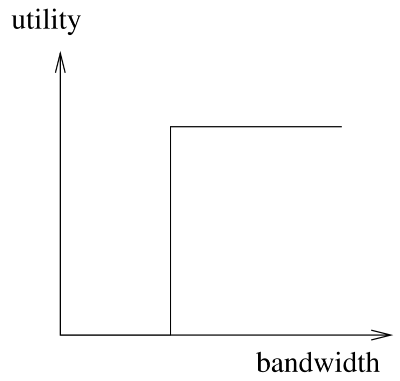

Figure 22.3: Hard Real-time Utility Function

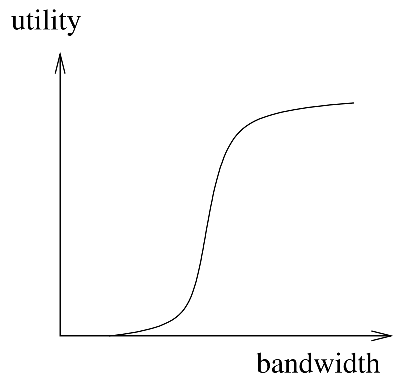

Figure 22.4: Soft Real-time Utility Function

Best-effort utility function is sort of linear, so the sum of the users' utilities will not be affected much in case of overload. But for real-time applications, the utility becomes zero when a user's bandwidth is below the threshold. Hence, the sum of the users' utilities will become very small in case of overload, since most of users actually get a below-the-threshold bandwidth.