CS294-1 On-line Computation & Network Algorithms	**Spring 1997**

Lecture 17: April 3

Lecturer: Danny Raz *Scribe: Chris Umans*

In this lecture we discuss "A Modular Analysis of Network Transmission Protocols."

17.1 Backround

We consider the problem of transmitting files through a network such as the Internet. The protocol is the method we use to determine when to send packets and which ones to send. Most Internet traffic uses the TCP protocol. Files are transmitted in chunks called packets. Packets can get lost, dropped, or delayed for unbounded amounts of time. The receiver can send ACKs and NACKs, which can also be dropped, but each one contains redundant information about the previous ones. We want a fast and efficient protocol and a good method for analyzing end-to-end packet protocols. Because packet loss does not fit a "nice" statistical model, we use an adversarial model for the analysis. We wish to have roughly in-order delivery, especially for such applications as audio and video delivered in streams.

17.2 Analysis

Our measure of how well the protocol performs measures the length of the contiguous prefix of received packets at a given time for algorithm A:

$$P^A(t,\theta) = \text{ length of contiguous prefix at time } t \text{ with network behavior } \theta$$

We use competitive analysis to compare this to the performance of the optimal algorithm, denoted $P^*(t,\theta)$. Since the optimal algorithm knows what the adversary is going to do, the value of $P^*(t,\theta)$ is always equal to the number of packets that successfully make it through the network at time t. Formally, the competitive ratio is:

$$R^A = \max_{\theta,t} \frac{P^*(t,\theta)}{P^A(t,\theta)}.$$

We want this to be as close to one as possible. When we obtain a ratio of one, we try to minimize the following value:

$$\Delta^A = \max_{\theta,t} P^*(t,\theta) - P^A(t,\theta).$$

There are two independent components to the algorithm: *data selection policy* ("what"), and *time scheduling policy* ("when"). The second says when a packet is to be sent, while the first decides which packet is to be sent. We say that θ includes the times scheduling policy as well as the

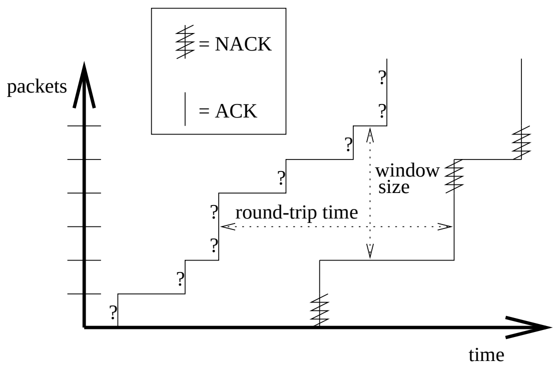

Figure 17.1: Basic diagram depicting θ

behavior of the network, and we fix it, so we just focus on the data selection policy independent of the other parameters.

17.3 Algorithms

One possibility is to send the packets in a round robin – that is, we send packets in order, cycling when we reach the end. If an ack is received, we remove that packet from the set that we keep sending; if a nack is received, then we immediately re-send that packet. Why is this not obviously optimal? Imagine a time scheduling policy that admits pairs of packets, and an adversary that kills the smaller numbered packet of each pair. Then we send $1, 2$; $1, 3$; $1, 4$; etc... and since 1 is never received, the ratio is infinite.

Do real networks behave this way? Maybe. Suppose two servers are both sending over a single link at the same time, causing congestion that results in the the first of every two packets getting lost. Or it might be that the receiver does not allocate enough buffers, drops the first packet, clears room by the time the second gets there, then releases buffers while waiting for the next, so the next one is dropped, etc...

Since θ is fixed, we have the situation depicted in figure 17.1 in which we must simply determine which packets to send at each send event, marked with a "?".

In the type of diagram in figure 17.1 we assume that ACKs and NACKs obey a FIFO rule. Two important parameters are the round-trip time (horizontal arrow), and the window size (vertical arrow). In this model, the window size reflects the uncertainty of the environment and is not necessarily a parameter of the algorithm.

Another simple algorithm is the fully redundant algorithm, which sends each successive packet repeatedly until an ACK is received for it. If the window size is B, then we have the picture in figure 17.2, and the ratio is B:

A third possibility is to use sliding windows. We set a window size C and do a round-robin on C packets, replacing old packets in the set with new ones as ACKS are received. However for a

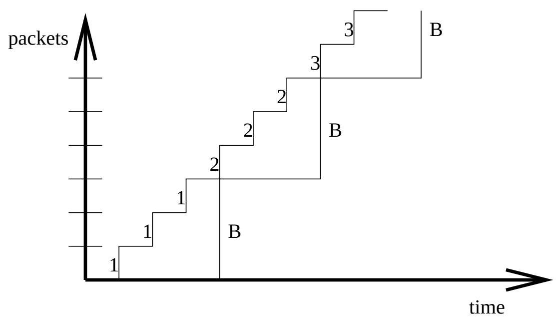

Figure 17.2: Fully redundant algorithm with window size B

window size of 2, and the type of network behavior described in conjunction with the basic round robin algorithm above, the ratio is infinite.

17.3.1 Optimal Algorithm

The optimal deterministic algorithm is a variant of the window protocols. In round i, we use a window of a fixed size k_i, which means that we limit the set of eligible packets to the next k_i unsent packets. As packets are acknowledged, they become ineligible for later transmission. At each send event, we sent the eligible packet that appears in the backlog the least number of times, breaking ties arbitrarily.

We now obtain an upper bound on the number of successful transmissions that this algorithm can achieve, which will be used to establish the performance of the optimal algorithm (whose prefix is the number of successful transmissions). Suppose that B is a bound on the window size, and suppose that $k_i > B$. Only the last B packets in round i might be sent more than once. Of those, the first $B/2$ elements can be sent at most twice; the next $B/4$ elements can be sent at most 4 times; etc...; and the last one can be sent at most B times.

So the number of successful transmissions of the last B packets in a given round is at most $B(\log B + 1)$, so the total number of successful transmissions in round i is at most $k_i + B \log B$.

Now suppose we choose $k_i = i$. At time t in round $i+1$ we have:

$$P^*(t, \theta) \leq \sum_{j=1}^{i+1} (i + B \log B) = \frac{(i+1)(i+2)}{2} + (i+1) B \log B$$

$$P^A(t, \theta) = \sum_{j=1}^{i} i$$

Formulating the ratio, we get:

$$R^A \leq \frac{\sum_{j=1}^{i+1} (i + B \log B)}{\sum_{j=1}^{i} i} = 1 + \frac{2(B \log B + 1)}{i}$$

Again using the fact that $P^*(t,\theta) \leq \frac{(i+1)(i+2)}{2} + (i+1)B\log B$, we get that:

$$R^A \leq 1 + \frac{4(B\log B)^{3/2}}{\sqrt{P^*(t,\theta)}}$$

and this tends toward 1 as the message length goes to infinity.

Finally, we note that $\Delta^A \to \infty$ as $i \to \infty$. We want to bound Δ but this cannot be done for any deterministic algorithm. The main result (which is covered in the next lecture) is that there exists a randomized algorithm that achieves $\Delta = B\log^2 B$ with high probability.

References

[1] Micah Adler, Yair Bartal, John W. Byers, Michael Luby, Danny Raz. *"A Modular Analysis of Network Transmission Protocols"*, To appear in Proc. of the 5th Israeli Symp. on Theory of Computing and Systems, June 1997.