Massachvsetts Institvte of Technology
Department of Electrical Engineering and Computer Science

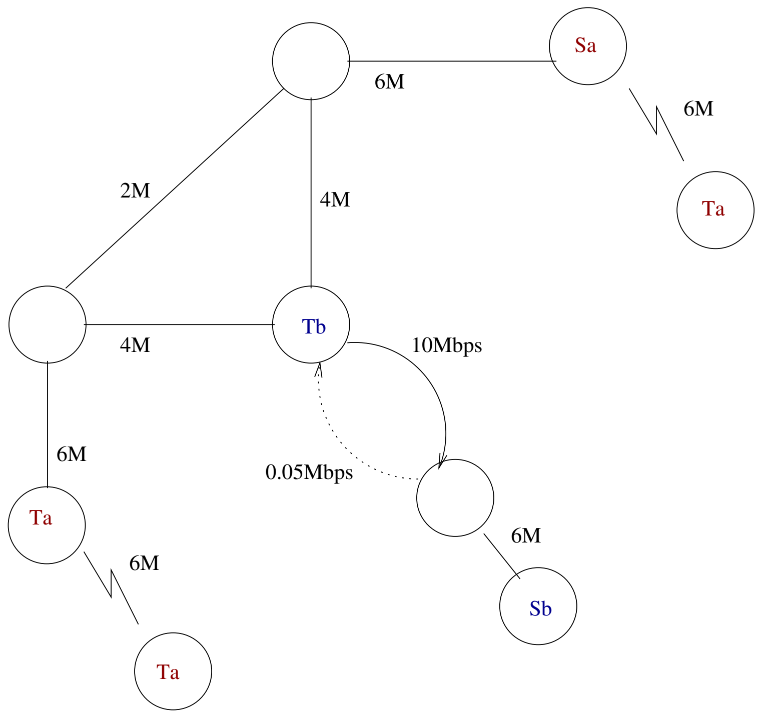
6.829 Fall 2001 **6.829 Competition: Fast Data Distribution** September 19 2001

---

1 Overview

This project is deceptively simple: Ship a set of files across a network as quickly as possible, from a set of sources to a set of destinations. Your protocols have to be "Internet-friendly" (or "TCP-friendly" as explained in class in Lecture 3) in that end-to-end congestion control must be done, but there's a lot of room for cool techniques to ship data across in the fastest amount of time. Who knows—a really innovative and good approach may be more widely applicable well beyond this class!

We will provide you with an emulated network setup, with computers connected in various ways with each other. The links will have varying amounts of bandwidth and delays; some of them may be simulated wireless links, where losses occur without any correlation to congestion. Some links may be highly *asymmetric* in their bandwidth or delay. The routing in the network may not be optimal, or there may be multiple paths from one point to another. Files may need to go from one point to another, or may need to go to many receivers. Links may change capacity *during* the experiment, or fail entirely for some periods of time. For example, the network below (which is *not necessarily* the topology on which your programs will be evaluated) is representative of the kind of features in the real network. Sources are S_{foo}, and sinks are T_{foo}.



Your job is to *reliably* and *correctly* transmit a large amount of data (in the form of files we generate) from the specified sources to the specified sinks as quickly as possible. Each source will have one large file. It will already be compressed, so (although it's a good idea in general) you shouldn't worry about compressing it!

2 Details

This section outlines the details of what's expected and the constraints you should work under.

2.1 Correctness

The first requirement of the project is (surprise!) *correctness*: You must get the entire file(s) across the network with no errors. We will verify this using a cryptographic checksum of the files. Similarly, you *must* use proper (e.g. TCP-friendly) congestion control on the links in the network unless otherwise specified. For certain special links (wireless, satellite links, modems, horsemen, carrier pigeons, etc.) we will relax this requirement and encourage you to use any protocol you wish. We will tell you in advance on which links you may use non-congestion-controlled (i.e., non-TCP-friendly) protocols.

2.2 Speed

We will examine both the *mean* completion time for the files at each node, and the worst-case (e.g. the time at which the entire process has finished delivering data to all sinks). If you think about this, you'll discover that a good strategy to optimize mean completion time across all receivers will in general be different from a strategy that optimizes the worst-case finish time across all sinks!

It's OK if your system does well on only one of the metrics, but a good system will do well at both.

2.3 Possible techniques

This is an incomplete list of things you might consider. It isn't an exhaustive list, but we think it's a very good starting point.

- Pipelining. This is probably essential. I.e., a system that sets up T TCP connections from a source to the T sinks isn't going to work well. In particular, if the source is connected using lower-bandwidth links than the rest of the network, this approach will likely lose. The way to approach the problem is to construct an *overlay network* of nodes to forward packets, tunneled (for example) across TCP connections between these overlay forwarder nodes.

  There's a whole area of research on this now, and you should look at the ideas in application-layer multicast. Two specific schemes to check out to get ideas are Overcast [4] and End-System Multicast [3]. You probably won't be able to use either of these schemes off the top, but ideas from these will be very useful. (You can get away with something simpler in terms of topology construction, most likely.)

- Wireless transport. Your topology will contain a wireless link, where all packet losses will be guaranteed (by us) to be due to packet corruption and not due to congestion. If you find that there's a path in the overlay that only includes wireless links, you'll be better off not running TCP over it. You can come up with a different protocol from TCP, or if you're particularly ambitious, adapt the Snoop protocol [2] to work in this network (although you'd have to port from BSD/OS to FreeBSD; it may take only a day or two to port, though!)

  Your new wireless protocol may use negative ACKs (NACKs) or it may use packet-level FEC, or both. To understand why TCP doesn't work so well over error-prone wireless, read [2].

- Asymmetric network transport. Read [1] (http://nms.lcs.mit.edu/~hari/papers/tcpasym-monet.ps.gz) to understand why TCP doesn't work very well across asymmetric networks. Protocols that use NACKs may work a lot better.

  For both the wireless and the asymmetric network cases, your new transport should run over UDP.

- Making sure TCP works well on the links you're using. This is critical.

- Adaptive application-layer routing (e.g., see http://nms.lcs.mit.edu/papers/ron-sosp2001.html) and multipath routing. You may find that routing packets along multiple paths between two nodes, and reconstructing them in order at the destination, is a great way to obtain greater application throughput. Investigate and develop this idea, if you find it's beneficial.

- Forward Error Correction (FEC). You may consider linking against a public-domain FEC library, or write your own. Check with us first before nailing down an implementation choice. Look at the properties of Reed-Solomon codes; they may be what you want but don't do it just because it sounds like a good thing!

- A good user interface (GUI). Would be nice to have a nice GUI to specify the sources, sinks, etc.

- Other cool things we haven't thought of. Good solutions, new research in this context, etc. are welcome. Also, use our suggestions as ideas, not gospel. A great solution may not, and need not, incorporate all the things we mentioned above.

- Do not worry about your solution scaling to a large number of nodes. The test network on which we'll run your protocols and system will be ≤ 12 nodes. We promise!

2.4 Rules

If you use any external software, you must run it past 6.829-staff@mit.edu first.

Use common sense and be nice: Please do *not* abuse the test network, or the real Internet. Don't try to hack root on these machines, change the operating environment or the time on the machines, modify the network parameters, or sabotage other students work. The machines in the test environment are part of the Utah Network Testbed, a research platform used by many people, not just us.

2.5 Environment

We will provide a test environment and an evaluation environment in which you can test your code. The test environment will be something like the diagram on the previous page; here, you can login and run your code to make sure it works, test the effectiveness of your optimizations, and so on. At the end of the semester, you will be given a chunk of time in which to run your code in the evaluation environment, and submit the results to us.

The environment consists of:

- 12 or fewer nodes running FreeBSD.

- Dummynet-based channel simulation to provide slower or lossy links, links with varying network conditions, links that go down and come up unannounced, so you'll have to discover their existence.

- We will tell you which links are wireless and which are asymmetric.

For your convenience, we will provide some software for monitoring links, and install some standard software. You *do not* have to use this software; we're simply providing it as a helping hand.

- The Resilient Overlay Networks group membership manager, prober, and performance database. This is a slightly smarter "ping" program from which you can extract some link stats. You'll probably find this very useful to discover path properties, to do performance-based packet routing.
- `traceroute, ping, etc.` the standard system utilities to determine link properties and topology.
- `ttcp` to run TCP throughput tests if you want.

Do not hack any kernels. Your software must run at user-level under FreeBSD using standard sockets.

2.6 Evaluation

- 5 minutes setup without network connectivity: Program installation, testing, etc.
- 10-15 minutes setup *with* network connectivity; you can do probing and topology discovery during this time, but no file transfers
- 15 minutes to transfer the files themselves
  (Times are subject to adjustment).
- A setuid program that checksums and timestamps the data, reporting to us that it's happy. Your programs should run the verifier as soon as the data has been shipped completely to one end point.
- We will examine both the *mean* completion time for the files at each node, and the worst-case (e.g. the time at which the entire process has finished delivering data to all sinks). If you think about this, you'll discover that a good strategy to optimize mean completion time across all receivers will in general be different from a strategy that optimizes the worst-case finish time across all sinks!

3 Logistics and Grading

This project should be done in teams of three. If you've decided to do this project, submit a 1-page proposal (see `http://nms.lcs.mit.edu/6.829/project-logistics.html`) with names, email addresses, and a short design summary of what you expect to build (specific design, but be brief). Also, include in the proposal any specific questions you have and any special requests for equipment, etc.

Grading will be based upon several things. The exact grading policy is not final yet.

- Project writeup: We'll add more detail here soon.
- Project test: We will run your code on the emulated network. Your code *must* successfully transfer the data across the network in a reasonable amount of time.

- Features and Performance: The better your project does, the better! Implementing more features to boost performance will result in a better grade. A good GUI will get you some bonus points. Clever algorithms win. This is an intentionally under-specified problem, and the range of good solutions is nearly unlimited.

At the end, you will turn in a conference-style paper clearly describing your design and implementation. We will hold a "6.829 bake-off" on December 7 2001 (Friday) to see how the different approaches compare! Free food!

Questions? `6.829-staff@mit.edu`

References

[1] BALAKRISHNAN, H., PADMANABHAN, V. N., AND KATZ, R. The Effects of Asymmetry on TCP Performance. In *Proc. ACM MOBICOM* (Sept. 1997).

[2] BALAKRISHNAN, H., SESHAN, S., AND KATZ, R. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks 1*, 4 (Dec. 1995).

[3] HUA CHU, Y., RAO, S. G., AND ZHANG, H. A case for end system multicast. In *Proceedings of ACM Sigmetrics* (Santa Clara, CA, 2000), pp. 1–12.

[4] JANOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND O'TOOLE JR., J. W. Overcast: Reliable multicasting with an overlay network. In *Proc. 4th USENIX OSDI* (San Diego, California, October 2000), pp. 197–212.