Adapting to Wide-Area Network Dynamics

Vern Paxson
EECS Division, University of California, Berkeley
vern@cs.berkeley.edu

Dissertation proposal / Qualifying exam

September 22, 1994

1 Motivation

Because of its efficient use of network resources and resilience in the presence of failures, packet-switching forms a key element of the global Internet's architecture [Cl88]. The gains of packet-switching come from sharing common resources among many network users. These gains in turn lead to the central problem of packet-switching: the network as seen by a particular user changes over the course of time, requiring the user to either adapt to the changes or suffer from a mismatch between the user's expectations of the network's behavior and the network's actual condition. Not only might users fail to use the network efficiently if they do not adapt appropriately, but if the network is overloaded and users do not scale back their use, the entire network can suffer "congestive collapse" and essentially cease functioning [Na84].

The principle technique used for adapting to changing network conditions is *flow control*, in which a network connection has associated with it either a *window* limiting the amount of outstanding data it can have in transit through the network, or a *rate* limiting the pace at which it injects data into the network [GK80, Ja90a]. The problem of adapting to changing network conditions then reduces to the question of what window size or transmission rate to use.

A key point is that the required degree and quality of adaptation increases with time. For example, the TCP/IP internetwork used the simplest possible window algorithm—a constant window size—for a number of years. Initially, this proved adequate, but as the network grew the rigidity of the algorithm eventually led to congestive collapse [Ja88]. At that point, Jacobson introduced an adaptive window technique called *slow start*, which is now required for all Internet hosts, and which has prevented further congestive collapse. In slow start, the window is initially set as small as feasibly possible. With each acknowledged packet, the window is then increased, since the acknowledgement indicates the network still has available capacity. A lost packet, on the other hand, indicates that the network is congested, and the window is cut back at that point. Slow start has a number of refinements (and problems) beyond this simple description, but the key point we wish to make is that it illustrates how adapting to changing network conditions can radically improve network performance.

In the future, adapting to changing network conditions will become both more vital and more difficult. It will become more vital because network conditions will become more and more variable, and it will become more difficult because network connections will have less and less time to adapt to changing conditions. We argue each point in turn.

Previous work has already shown that there is no typical traffic profile; rather, the "mix" of which application protocols dominate the traffic at a particular site varies greatly from site to site [DJCME92, Pa94a]. Furthermore, the traffic mix at a particular site varies considerably over time, and the characteristics of individual application protocols vary significantly from site to site [Pa94a]. In addition, both overall network use and network use at individual sites has exhibited sustained exponential growth over many years[1]. New applications such as X11, World-Wide Web, and audio/video traffic can exhibit explosive growth [Pa94b]. Finally, the recent success of "self-similar" stochastic models of network traffic [LTWW94, PF94] matches the observation that aggregate network traffic shows large variations across all time scales, ranging from milliseconds to months, even in the presence of high degrees of multiplexing [FL91].

These findings all dramatize how difficult it is to devise solutions to networking problems unless one has some knowledge about the current state of the network. As network use grows and becomes more heterogeneous, these problems become more and more acute.

Another dimension to the problem is that network bandwidth is increasing extremely rapidly. For example, over a six year period the speed of the NSFNET backbone links grew from 56 Kbps to 45 Mbps[2], increasing 200%/year [CBP94]. As bandwidth increases, it becomes harder for connections to fully utilize the available bandwidth without attempting to overuse it. If a connection underutilizes the available bandwidth, it both suffers poor performance, and its traffic becomes bursty. In the absence of fair queueing schemes [DKS90], such burstiness then spreads to other, previously well-behaved connections. On the other hand, if a connection overuses the available bandwidth, well-behaved connections again are affected, now by incurring unwarranted delays or losses.

Thus, in the future it will be increasingly important for network connections to discover and adapt to the current network conditions.

Rapidly increasing bandwidth leads to another important effect. Because the lower bound of end-to-end delay in networks is limited by the (constant) speed of light, as bandwidth increases, the network carrying capacity (bandwidth-delay product) increases too. This increase is particularly relevant for wide-area network traffic, which is our scope of interest in this proposal. Larger bandwidth-delay products mean that network connections will have more data "in flight". But our previous work characterizing TCP connections [Pa94a] showed that the median size of TCP connections is either *not* increasing over time, or only increasing slowly. Since the size of the network "pipe" is increasing but connection sizes are not, con-

[1] For example, in the last five years the Internet has grown from 130,000 hosts to 3,200,000 hosts, a rate of 90%/year.

[2] And this will increase to 155 Mbps by 1995.

nections will take fewer and fewer round-trip times to transmit their data. While this is a performance boon from the user's perspective, it makes adapting to network conditions more difficult: because it takes a round-trip time to discover anything about network conditions, future connections will have fewer and fewer opportunities to adapt. Ultimately, *most connections will fit into a single round-trip time*, and thus will have *no* opportunity to adapt the later part of their transmission based on observations of how the first part of their transmission fared. Because of this impending "feedback crunch", with high-speed networks it will become vital to estimate as much as possible about current network conditions based on the performance of *past* connections.

These arguments motivate our dissertation proposal, which has three elements. First, develop techniques for inferring as much as possible about network conditions based on observing traffic patterns. Second, demonstrate the effectiveness of these techniques by showing that they can be used by a network connection to optimize its present performance based on analysis of its past performance. Third, determine how a new network connection can best utilize information gathered by past connections to do the same sort of optimization.

In the remainder of this paper we expand on each of these elements in detail. While in this proposal we restrict ourselves to adaptive techniques available to network transport endpoints, the same principles (with some alteration to mechanisms) apply to internal network routers as well.

2 Sources of Information About the State of the Network

Today's transport protocols use several sources of information in order to adapt to network conditions:

- Reliable communication requires some form of *acknowledgements* ("acks"). These are messages sent by the data receiver to the data sender indicating either which data packets have arrived successfully (positive acks), which are missing (negative), or a combination of the two (selective).
- Related to acks, the *loss* of a packet is interpreted as a sign of possible network congestion.
- The *round-trip time* ("RTT") is estimated by timing the interval between transmitting a selected packet and receiving its acknowledgement. An estimate of an upper-bound on the RTT is used in setting retransmission timers in order to determine that a packet has been lost (in the absence of negative acks).
- *Duplicate* (repeated) acks are used as a form of negative ack, to spur retransmission of an apparently missing packet.
- The *advertised receiver window* indicates the buffer available at the receiver, which limits the amount of unacknowledged data the sender can have in flight.
- The time structure of the received acks forms a *self-clocking* mechanism, used to implicitly adapt to current network capacity (see [Ja88] for a nice illustration). We will return to this notion in greater depth below.
- Explicit *congestion-notification signals* are sent to the data sender by routers internal to the network, to indicate that the network is heavily loaded.

These information sources are all used by current TCP implementations.[3] Other reliable transport protocols such as TP4, XTP, VMTP, SNR, NETBLT, Delta-t and MSP [CLZ87, WM87, CW89, DDKMRW90, NRS90, BD92, PS93] do not incorporate any additional information sources, except as we discuss shortly.

We make a distinction here between *information sources* and *adaptation algorithms*. There are, for example, many proposed modifications to TCP to alter how it responds in light of the information it gathers. Our emphasis is not on new adaptation algorithms utilizing existing information, but on acquiring new sources of information, and then investigating how these new sources might in turn lead to new adaptive algorithms.

The information sources discussed above allow a transport endpoint to answer several types of questions: Did the data arrive successfully? Is the network heavily congested? How long to wait before deciding a packet was lost and needs retransmission? Is there a hole in the data "pipe"? Does the receiver have adequate buffer for more data?

There are a number of additional questions we'd like to be able to answer, which require additional information about the network's state:

- *Is there a pipelining mismatch?* Fully utilizing the network requires transmitting data at a rate matching that of the slowest link. This link is referred to as the "bottleneck". While the bottleneck may be the speed of the sending or receiving processor, usually the limit is the transmission speed of one of the elements of the network path. Assuming that the network is indeed the bottleneck, then this question is equivalent to asking, Is the window size (or, equivalently, sending rate) too large, too small, or correctly matched to the capacity of the network? To answer this question, we need to know the *bottleneck bandwidth*.
- *Is congestion building up?* Rather than waiting for packet drops as indications of congestion, we would like to recognize congestion just as it begins, before the congested gateway's queues become completely full. Answering this question requires information regarding variation in queueing delays along the network path.
- *How much are we adding to queueing?* If our packets are waiting in queues inside the network, then we are buying a modest decrease in latency at the cost of increasing the network load. Different applications have different latency/throughput requirements, and thus different tolerances for how much queueing they incur, but they can all benefit from information regarding their contribution to overall network load. To determine whether we are adding to queueing requires comparing the predicted RTT change for a given window or rate adjustment vs. the observed change. Predicting the RTT change in turn requires estimating the bottleneck bandwidth and the propagation time for the network path.
- *Is self-clocking working correctly?* As mentioned above, TCP relies on self-clocking to pace its packets at the correct output rate. Self-clocking fails, though, in the presence of *ack compression*, which occurs when the time structure of

[3] Though selective acks are not widely implemented, and the only congestion-notification signal TCP responds to is the now-deprecated "source quench" ICMP message [St94, p. 161]. Recently, several refined congestion-notification signals have been proposed [RJ90, FJ93].

acknowledgements is destroyed by adverse network dynamics. We discuss the phenomenon of ack compression, along with ways that an endpoint can detect it, in Section 5 below.

One recent additional source of information is the variation of round-trip time. Adaptation algorithms using this information are based on the following insight: If increasing a connection's window size leads to an increase in RTT, then the connection is contributing to queueing (and hence congestion) [Ja89]. Jain's work is theoretical. Wang and Crowcroft have since proposed a scheme for incorporating RTT variation into TCP to avoid periodic packet loss due to excessive load [WC92], and to avoid contributing to queueing [WC91]. Along similar lines, Sanghi and Agrawala have devised DTP, a transport protocol similar to TCP but which uses RTT measurements of past packets to time the sending of new packets [SA91]. More recently, the authors of the "Vegas" modifications to TCP [BOP94] propose using increases in RTT to assure that each TCP connection acquires a minimum of α buffers at the bottleneck queue, and a maximum of β buffers, where $0 < \alpha < \beta$ (though this approach appears too aggressive because it is guaranteed to add load to an already-congested network).

The thrust of these recent approaches has been to use RTT variation to answer the second and third questions posed above, as well as indirectly addressing the question of bottleneck bandwidth. An important point, however, is that to reliably interpret RTT variation, one must first answer the final question—is self-clocking working? To our knowledge, detection of ack compression has not been incorporated into any transport protocol mechanisms. We return to this point in Section 5.

If one can provide accurate estimates of the quantities needed to answer these questions, then one has the fundamental information needed to implement a variety of theoretical schemes for optimizing network performance [Ja81, Ja89, MS90, SA91]. We now turn to the specifics of our research proposal, the foundation of which is to refine existing methods and develop new methods for estimating these quantities, and then to build on the new information sources to optimize wide-area network performance.

3 Overview of the Proposal

The general goal of our proposed research is to first develop methods of gathering as much information as possible about a network's current state, and then to show the utility of this information by applying it to a particular networking problem. Our emphasis will be on how information can be gathered by a transport connection endpoint operating on the Internet, but it is important that we also keep in mind how the techniques might be modified for use by network routers, so we can in the future perhaps move the adapting entity into the network rather than relying on endpoints.[4] Similarly, since one of our goals is to develop techniques applicable to the Internet, we need to also keep in mind the degree of compatibility our techniques allow: whether they can be done with no modifications to existing protocols; by introducing new protocol options; by modifying routers; or only by modifying the protocols. Our belief is that much of the information we want to gather can be acquired without introducing incompatibilities; we cover this further in Section 4.

We propose conducting a series of three experiments:

1. Using both existing and newly-developed techniques to gather information about network conditions, characterize the dynamics of Internet queueing, bottlenecks, losses, quiescence, congestion, and ack compression. The methodology for the experiment is to use TCP packet streams from and to a variety of Internet sites as probes of the network's state.
2. Apply these characterizations to the "bandwidth discovery" problem, in which a connection infers the bandwidth available to it by observing its past performance and adjusting its sending rate accordingly.
3. Assess the degree to which past information regarding bandwidth discovered by previous connections can then be incorporated into optimizing new network connections.

We discuss these experiments in Sections 6, 7, and 8 below.

4 Acks as RADAR Echoes

A key claim of our proposal is that a large amount of information about network conditions can be inferred by a connection endpoint, without any assistance from the routers along the connection's path. In this section we present the basis for this claim.

The basic operation of a sliding-window protocol such as TCP is as follows:

1. The sender transmits packet i at time t_i.
2. The packet incurs a delay d_i traversing the network, arriving at the receiver at time $t_i + d_i$.
3. After some amount of processing p_i, the receiver acknowledges the packet at time $t_i + d_i + p_i$.
4. The acknowledgement takes time a_i to traverse the network, arriving back at the sender at time $t_i + d_i + p_i + a_i$.
5. With the new acknowledgement the window slides forward, allowing the sender to transmit packet $i + w$ after a processing delay p'_i, where w is the window size.

The round trip time for packet i is then

$$\text{RTT}_i = d_i + p_i + a_i.$$

Acknowledgements are small enough that their own contribution to queueing on the return path is negligible. If, then, the acknowledgements do not encounter any significant queueing, the quantity a_i will be constant.[5] Furthermore, if the receiver generates acknowledgements promptly (more on this below), then p_i will be small, and we can approximate $p_i + a_i$ as a constant γ. This then gives us the relationship:

$$\text{RTT}_i \approx d_i + \gamma\,.$$

Thus, apart from a constant offset γ, the quantity RTT_i, which we can measure, also gives us the one-way delay d_i. We thus can construct a time series sample path $\{\text{RTT}_i\}_{i=1,2,\ldots}$, whose structure can in turn tell us a great deal about the condition of the links over which the packets traverse. In this sense, we can view a packet's acknowledgement as a "RADAR echo": we bounce a packet off a distant object (the receiver), and the time structure of the echoes reflects the distortions introduced by the network internals.

[4] In particular, if we rely on adaptive techniques to ensure that the network remains stable, then it is administratively far easier to require routers to run particular adaptive algorithms than endpoints.

[5] We test this assumption in Exp. #1 (Section 6).

In order both to gauge how good an approximation it is to hold γ constant and to then interpret what d_i tells us about network conditions, we need to analyze the different sources of delay that a packet and its acknowledgement might encounter. We can categorize these delays as follows:

- *transmission*: time required for the packet to traverse the physical links. For a path with propagation delay P and bandwidth B bytes/second, a packet of size b bytes will have a transmission time of $T = P + b\,/\,B$ seconds.
- *bandwidth*: because of the relationship described in the previous item between bandwidth, transmission delay, and propagation delay, we will consider bandwidth as a delay element.
- *media access*: for example, with an FDDI ring, the time waiting for the token, or, with an Ethernet, until the packet is transmitted without collision.
- *receiver protocol processing*: time required for interrupt scheduling, interpreting the packet, creating the corresponding acknowledgement, and beginning its transmission. Except for contributions due to the next two items, this overhead is equal to the time p_i discussed above.
- *delayed acks*: many TCP implementations do not immediately acknowledge received data, but instead schedule an acknowledgement for later delivery, in the hope that more data will arrive in the interim and the acknowledgements merged. For Berkeley-derived TCP implementations, for example, this delay can introduce a lull of 0-200 msec [WS94]. This extra delay contributes to p_i.
- *receiving process can't keep up*: if the receiving process can't consume the incoming data fast enough, it will lead to delays: either an increase in p_i, if acknowledgements are not sent until the receiving process frees up some buffer space, or an increase in p'_i, if the sender is prevented from sending due to the window closing.
- *sending process can't keep up*: if the sending process can't provide data fast enough, then p'_i will inflate due to waiting for new data to accumulate.
- *router processing*: time required for the intermediary routers to forward the packet once it reaches the front of the send queue.
- *queueing*: the remainder of delay is due to time spent by the packet waiting in queues at the intermediary routers.

By observing the structure of the time series $\{\mathrm{RTT}_i\}$, a connection endpoint can assess or account for these delay sources as follows:

- *transmission*: the minimum RTT observed gives an upper bound on the transmission delay.[6]
- *bandwidth*: we can estimate the bottleneck bandwidth using a variant of the "packet pair" technique [Ke91], which works as follows. We transmit two packets n and $n+1$ back-to-back, separated by a small amount of time ϵ. Each packet is of size b bytes. If the receiver and the network are unloaded, then the acknowledgement echoes will arrive at times:

$$\begin{aligned} e_n &= t_n + d_n + p_n + a_n \\ e_{n+1} &= t_{n+1} + d_{n+1} + p_{n+1} + a_{n+1} \\ &= t_n + \epsilon + d_{n+1} + p_{n+1} + a_{n+1} \\ &= t_n + \epsilon + d_{n+1} + p_n + a_n \end{aligned}$$

  where the last equality comes from the assumption that the receiver and the network are unloaded, hence $p_{n+1} = p_n$ and $a_{n+1} = a_n$.

  Consider the quantity $d_{n+1} - d_n$, the difference in propagation time experienced by the two packets. Since the network is unloaded, this difference is ϵ plus the amount of time packet $n+1$ must wait while packet n propagates across the bottleneck link. The key here, however, is that we do not have to wait for packet n to propagate entirely across the link, but only for its last bit to begin propagating, after which we are free to send the first bit of packet $n+1$.

  After packet n's arrival at the (unloaded) bottleneck, it takes time $b\,/\,B$ for packet n's last bit to begin propagating. Packet $n+1$ arrives at the bottleneck at time ϵ after packet n arrived, so it must wait for a time $b\,/\,B - \epsilon$ until it can begin transmission (assuming $\epsilon < b\,/\,B$).

  Once both packets have traversed the bottleneck, the resulting spacing between them is preserved. This gives us the following relationship:

$$\begin{aligned} b\,/\,B - \epsilon &= d_{n+1} - d_n \\ \hat{B} &= \frac{b}{e_{n+1} - e_n}. \end{aligned}$$

  The analysis of packet pair in [Ke91] requires that the network uses fair queueing internally, but the principle still holds for FIFO queueing if the network is unloaded. In particular, the same principle forms the basis of the self-clocking discussed in [Ja88]. Self-clocking leads to packets departing the sender in "flights" whose size is equal to the window size. By applying a packet pair-style analysis to these flights, we can then derive a more robust estimate of the bottleneck bandwidth (one that allows us to relax the restriction that the network be fully unloaded).
- *media access*: If the network path includes routers on multiple-access links, then media access for those links will introduce variable delay. For example, for a large, heavily-loaded FDDI ring configured per Jain's recommendations [Ja90b], packets transmitted in asynchronous mode can suffer delays on the order of 8 msec. Large, heavily-loaded Ethernets can incur comparable delays [BMK88].

  Thus, media-access delay could add a large element of noise to the round-trip times, depending on the network configuration. On the plus side, we will interpret this noise as being due to queueing (see below); since it *is* due to heavy load, this interpretation is not too far from the mark.
- *receiver protocol processing*: while we have no way to directly assess this delay element, we have reason to hope it is small, since for example Clark et al. have shown that TCP processing requires only a couple hundred machine instructions [CJRS89]. We accommodate this delay as a hopefully-small noise element.

[6] Except if the network path changes. We can detect route changes to some degree by noting the arrival of out-of-sequence IP packets, or a change in the TTL hop-count in the IP header.

- *delayed acks*: while these delays can be quite large, they are easily detectable for Berkeley-derived TCP implementations, which generate immediate acks if they can acknowledge at least two TCP segments. Thus an acknowledgement for a single segment corresponds to a delayed ack and we discard the associated RTT as untrustworthy, while acks for two or more segments reflect no additional delay when computing the RTT for the last segment acknowledged. In practice, this approach means we can perform our time-series analysis only for every other segment, reducing the information we have available by half. This reduction is unfortunate but not drastic.

- *receiving process can't keep up*: if the process consuming the data fails to do so quickly enough, it can manifest delays in two different ways. If the slow-down is temporary then, depending on the transport protocol implementation, the receiver processing time p_i may increase as the transport endpoint delays acknowledging data.[7] If the slow-down is permanent, then the window will eventually close, which we interpret as meaning that the receiving process is in general the bottleneck (and hence increasing our sending window or rate can't gain us anything).

- *sending process can't keep up*: the sending transport endpoint immediately knows if the sending process can't keep up (because it sees that the window allows more data to be sent, but the data's not ready). In this case it again makes no sense to increase the sending window or rate.

- *router processing*: we hope that this delay is quite small, since router fast paths are usually highly optimized. It is possible, though, for very large delays to occur, for example due to global synchronization of routing messages [FJ94]. In general such conditions are detectable since one of their salient characteristics is their periodicity, but the intervals are often on the order of tens of seconds, so during the life of a single transport connection it may be difficult to tell when some of its packets incur abnormal delay due to this effect.

- *queueing*: as we have accounted for all of the other sources of delay or argued that they can usually be modeled as small noise terms, we can interpret the unaccounted-for delay as due to queueing. Let the estimate of queueing delay for the ith packet be $\hat{Q}_i$. Since we know the estimated bottleneck bandwidth $\hat{B}$, if we assumed that the bulk of the queueing delay occurs at the bottleneck link, we can assess $\hat{S}_i$, the estimated queue size encountered by packet i, as:

$$\hat{S}_i = \frac{\hat{Q}_i \hat{B}}{b} \text{ packets.}$$

  An important point here, though, is that if the bottleneck server suffers from *persistent* queueing (i.e., its output queue never empties), then our estimated minimum RTT (first item above) will be too high, which will result in underestimating $\hat{Q}_i$.

Thus we claim that from the time series $\{\text{RTT}_i\}$ we can estimate all the quantities necessary to characterize the network's dynamic state.

[7] This is the case with Berkeley-derived TCP implementations, which delay acknowledging data until the receiving process has consumed two segments worth of queued data [WS94].

5 Ack Compression: Corrupted Time Structure

The techniques given in the previous section for estimating the network's state rely heavily on the assumption that the spacing of acknowledgement arrivals reflects the spacing of data packet arrivals seen by the receiver. That is, early in our analysis we made the assumption that a_i is constant. The reason to believe a_i might be constant is because acks are sent already spaced out to reflect the bottleneck bandwidth (the "self-clocking" effect), so only large perturbations will significantly alter their spacing.

It is unlikely that acks will become more widely spaced as they traverse the network, as this requires a large intervening burst of traffic between adjacent acks. A more serious possibility, however, is that the spacing between a window's worth of acknowledgements (or some subset) might become condensed. This can happen if two adjacent acks each have to wait in the same router's queue and little or no intervening traffic arrives between them.

Ack compression was predicted from theory and simulation by Zhang, Shenker and Clark [ZSC91], and subsequently measured in the Internet by Mogul [Mo92]. Mogul detected ack compression as follows. First, consider two acknowledgements, A_1 and A_2, which differ in arrival time by a quantity δ_t, and in the number of bytes they acknowledge by δ_b. Define the bandwidth associated with the two acks as $\mathcal{B} = \delta_b/\delta_t$. Compute $\bar{\mathcal{B}}$, the median over all consecutive acknowledgements. Next, group acknowledgements into "ack windows" of a specified size and compute the same bandwidth figure between the first and last acks in the window. If this bandwidth exceeds $\bar{\mathcal{B}}$ by a specified ratio, then mark the window as compressed.

Using this definition, Mogul's analysis of the ack compression events found that compression was correlated with packet loss but considerably more rare. His study was limited, however, to a single 5-hour traffic trace.

We propose using a different definition of ack compression, one which explicitly preserves the notion of corrupted time structure. If for a given flight of data packets the bandwidth of the acks exceeds the original transmission bandwidth, then we consider those acks compressed. This definition has a significant advantage over Mogul's: instead of requiring a threshold based on the connection's entire past behavior plus a somewhat arbitrary cutoff ratio, we can detect ack compression using just the information associated with the current flight of packets, which we need anyway in order to construct and interpret the $\{\text{RTT}_i\}$ time series.

If ack compression proves rare, then detecting it remains a minor problem. If, however, it proves relatively common, then its impact on our studies depends on whether the compression events are large (gross compression, or sustained during much of a connection's lifetime) or small. If large, then ack compression poses a serious problem for the rest of our studies, and we will need to devise schemes to circumvent the problem. For example, we could compensate for ack compression by modifying receivers to timestamp data packets upon arrival and include these times in the corresponding acknowledgements; or, we could shift the information-gathering from the sender to the receiver, which could then control the sender either by modifying the offered receive window or by controlling the pace at which it issues acknowledgements.

In any case, characterizing the qualities of ack compression in the Internet is one of our top priorities, to be addressed in our first proposed experiment.

6 Exp. #1: Characterizing Internet Dynamics

The goal of our first experiment is to test the viability of our endpoint information-gathering techniques by using them for a large-scale study to characterize the dynamics of the Internet. Several similar studies have been made before, most characterizing the Internet on time scales larger than those necessary for today's connections. Kleinrock studied many different aspects of the the early ARPANET's behavior, emphasizing time scales ranging from hours to days [Kl76]. Mills conducted a series of "ping" experiments to evaluate the effectiveness of the TCP retransmission-timeout algorithm for traffic to a large number of ARPANET sites [Mi83]. Heimlich [He90] and Claffy et al. [CPB93a] characterized Internet backbone traffic, though not on the scale of individual connections.

More recently, Bolot conducted an intriguing experiment similar in many regards to what we propose [Bo93]. He used as probes UDP packets sent at fixed intervals and echoed by remote Internet sites. He then analyzed the variation of the corresponding $\{\mathrm{RTT}_i\}$ time series in order to assess bottleneck bandwidth and correlations between packet losses/delays as he varied the probe interval. He found that probe packet losses were only weakly correlated, provided that the interval between probes was large enough that the probe packet itself was not adding to queueing.

Claffy et al. conducted a similar study using ICMP echo request packets [CPB93b]. Their main goal, though, was to demonstrate that latency in one direction along a path often differs from that along the return direction.

The first step of our experiment is to gather as many participating Internet sites as possible, which we plan to do by appealing to fellow networking researchers (a number of whom have already indicated interest). A site can participate in two ways: by giving permission for us to send packets to the TCP `discard` port of one of the site's computers, or by allowing us to originate TCP probes from the site and trace them using a packet-capture tool such as *tcpdump*. The first level of participation involves no administrative work on the part of the site, and we anticipate that recruiting such sites will prove very easy. The other level of participation requires some administrative effort, but from our discussions with fellow researchers we believe we will be able to find enough participants at this level to make our experiment viable.

The measurement part of the experiment consists of picking (at Poisson intervals) a pair of hosts from our collection, between which we then conduct a series of probes. We first use the *traceroute* facility to identify the path between the sender and the receiver. After waiting a small (Poisson) time, we then establish a TCP connection from the sender to the receiver's `discard` port, send a number of packets (100-500 KB), and trace the time structure of the packets and their acknowledgements by running a packet-capture tool situated as close as possible to the sender. Ideally, we will also run a packet-capture tool at the receiver. When the transfer is complete, we use *traceroute* again to detect whether the route changed during the transfer. We repeat these experiments as many times as possible, so in the subsequent analysis we will have enough data to estimate frequencies of rare events, stability of measurements over time, variability of measurements across network paths, and noise distributions.

The key differences between our approach and Bolot's are the use of TCP packet streams rather than regularly and more widely spaced UDP packets, giving us finer time resolution at a cost of more complicated dynamics due to contributing to network load; our long intervals between probing periods, instead of continuously sending probes, which is necessary to avoid loading the network unduly; and the use of many Internet sites, so we can make "large number" arguments concerning the generality of our results.

We plan to analyze the resulting data to characterize a number of properties of the Internet:

- *packet loss*: What sort of patterns do we find in packet losses? Periodic losses may indicate global synchronization [FJ94], which can often be fixed by injecting randomness into the synchronized system. Correlated losses affect congestion control strategies, and are also particularly relevant to transmitting audio [Bo93].
- *ack compression*: How prevalent is it? Once it occurs, how long does it continue? Is the method proposed in Section 5 adequate for detecting it?

  We can answer these questions by tracing probe streams at both the sender and the receiver, which allows us to determine one-way latencies and thus the fidelity with which the acknowledgements preserve the timing of data packet arrivals at the receiver. That is, by tracing at both ends we can evaluate the soundness of the assumption made in Section 4 that the acknowledgement latency a_i is well-approximated as constant.
- *queue sizes*: How quickly do they vary? How large do they get? Is the variation consistent with self-similar models of wide-area traffic?
- *bottleneck bandwidth*: Do we consistently observe the same bottleneck bandwidth for the same path? How often do different paths that share some common elements also share the same bottleneck?
- *quiescence*: How often is a given path unloaded? Can we reliably detect quiescence?
- *onset of congestion*: Can we predict when congestion is building up? Can we predict incipient packet loss? (For example, Fowler and Leland found that for local-area traffic a crescendo of queueing spikes often preceded packet loss [FL91].)
- *contribution to queueing*: Can we accurately assess our own impact on queueing along the network path?
- *frequency of routing changes*: Most adaptive algorithms rely on the assumption that Internet routes change only rarely. Over what time scales do we see changes? How long do they persist?

An important issue in trying to answer these questions by tracing traffic at an endpoint is how to calibrate our measurement methodology so we have confidence in its correctness. Two mechanisms are available to us. The first is to use simulation: a second party sets up a multi-source simulation of wide-area traffic and gives us a trace of packet send times and acknowledgement receive times as seen at one of the endpoints. We then attempt to infer the internal characteristics of the network and compare our estimates with the network state as recorded by the simulator. The other calibration mechanism is to occasionally introduce cross-traffic of known characteristics (such as another `discard` connection, but to and from different hosts) and testing whether we can deduce its presence

based on differences between traces when the traffic is present and when it is absent.

This experiment is the most fundamental of the three we propose, and the other two might be substantially modified based on this experiment's results. It is possible that this experiment will reveal that assessing the network's state by tracing at an endpoint simply isn't viable due to excessive noise. If so, then that failure will provide the basis for another line of research, namely how the network can either provide more explicit information to endpoints, or adapt internally if it can't rely on endpoints to adapt intelligently.

7 Exp. #2: Bandwidth Discovery

Our discussion of the remaining two experiments is necessarily more brief, since their exact form will depend heavily on the results of the first experiment. We outline them in this and the next section, pointing out where they rely on findings from Exp. #1.

While the use of adaptive windows (or rates) in today's transport protocols is often with an eye to managing congestion by keeping the window from growing too large, a related problem is how to open up the window quickly when it is too small. We refer to this as the "bandwidth discovery" problem. Most bandwidth discovery schemes are based on opening the window a certain amount, observing the effect on latency and throughput compared to what is expected, and if the network still has capacity, repeating this cycle. The fundamental problem with this approach is that it involves several round-trip times (one per "observing the effect"), even if the window is opened exponentially fast, as is often the case.

We propose investigating a more aggressive approach. *If* Exp. #1 reveals that the network is often quiescent (unloaded), and *if* the experiment shows that we can reliably assess the bottleneck bandwidth, then a connection could in principle send a group of packets to determine these things, and upon confirmation open its window as far as the bottleneck bandwidth allows. If successful, then the connection discovers and begins to utilize the available bandwidth in a single round-trip time.

There are a number of issues that must be dealt with:

- It may turn out that while the network is generally not completely quiescent, it is often lightly loaded. If so, we need to evaluate what fraction of the available bandwidth we should attempt to seize given the degree of perceived load.
- A less aggressive version of this approach is to use the inferred bottleneck bandwidth as a new upper bound on window size (replacing the TCP "slow-start threshold").
- A still less aggressive approach is to use the inferred bandwidth only as a ceiling: we lower the TCP slow-start threshold to reflect the inferred bandwidth, if less, but we do not raise the threshold if the inferred bandwidth is more. This approach results in a TCP more conservative than those in use today.
- If we introduce a jump in the window size then we will lack corresponding acknowledgement packets to clock out the newly-liberated data packets. Thus we may need to introduce additional timers in order to clock out packets at a steady pace. Doing so could gain the benefits of both window protocols and rate-controlled protocols: steady traffic but with automatic scaling back if the network becomes loaded and acknowledgements are delayed.
- For such an algorithm to work in the actual Internet, we need to show that TCP's using the algorithm do not unfairly starve out TCP's without it, and that if all TCP's used the algorithm, the network would remain stable.

We could evaluate the effectiveness of our bandwidth discovery algorithm using the same experimental framework as before. At Poisson intervals we would establish TCP connections between different pairs of sources and receivers. Some of the sources would run the modified TCP and others the unmodified (control) TCP. We would then evaluate the throughput achieved by each type of TCP vs. the losses it incurred vs. its contribution to queueing. We would hope to find that the modified TCP increases throughput without additional loss or queueing.

8 Exp. #3: Utilizing Past Knowledge

We argued previously that as bandwidth-delay products increase, connections have fewer and fewer round-trip times over which to adapt to the current network conditions. In particular, small connections (which are very common) will eventually have *no* round-trip times available for adapting unless they are willing to incur otherwise unnecessary delays. One approach for providing such connections with the information they need to appropriately pace their transmission is to attempt to incorporate information gained by previous connections.

A simple example would be that each time a host begins a new connection, it checks to see the bottleneck bandwidth measured most recently and simply uses that. Thus, the TCP uses the bandwidth discovery algorithm evaluated by Exp. #2, but the bandwidth was actually discovered by an earlier *connection* rather than earlier packets of the current connection. As the new connection's acknowledgements arrive, the host applies the bandwidth discovery algorithm to find out what bandwidth it *should* have used, and that becomes the new remembered bottleneck bandwidth.[8]

Using past information raises a number of issues:

- The key question is how accurately do past measurements predict current conditions. This question will be answered in part by Exp. #1, which will characterize the stability over time of information such as bottleneck bandwidth and degree of network load.
- In general one would expect that information becomes less reliable the older it is. Can we associate a level of reliability with information based on its age so we can still make (appropriately cautious) use of it?
- When a new connection terminates, how do we incorporate its measurements with those we already have?
- To what degree can we use information acquired for connections to destinations *different* from the one we now wish to send to? Such information might still apply to our new connection if the two connections shared a sufficient portion of the network path. Exp. #1 will provide the data necessary for beginning to investigate this question.

We evaluate the effectiveness of using past knowledge with the same framework as for Exp. #2: we compare the throughput, loss,

[8]Modern TCP implementations already do this to a limited degree: the Net/3 TCP code keeps a per-route cache of route characteristics such as RTT, variance of RTT, and slow-start threshold [WS94].

and queueing characteristics of TCP's using different information-synthesis algorithms with those of control TCP's that do not synthesize any past information.

9 Summary

Since wide-area networks are increasing in speed and carrying capacity, and since network traffic varies enormously over time and from site to site, it is increasingly important that network connections adapt as best as they can to current network conditions. Adapting to changing conditions requires information about the network's dynamics: available capacity, degree of queueing, and impending congestion. We argue that by judiciously examining the time structure of the acknowledgement echoes received for previously transmitted data packets, a network endpoint can estimate to a large degree the conditions the data packets must have encountered during their trip through the network. We point out that the main impediment to such an endpoint-oriented approach is the phenomenon of "ack compression", in which the time structure of the acknowledgement echoes is destroyed due to heavy queueing on the return path. To deal with this potential problem, we suggest a new technique for identifying ack compression, as well as alternatives should the problem prove endemic to the Internet.

We then propose a series of three experiments to explore the quality of information an endpoint can gather concerning network conditions. In the first experiment, we use streams of TCP packets between a large number of Internet sites as probes of the Internet's state. We can then analyze traces of these streams to characterize the Internet's dynamics and to assess how accurate our information-gathering techniques prove in practice. We include in our experiment methodology two mechanisms for calibrating the accuracy of our results.

Our second experiment applies gathered information regarding network quiescence and available bandwidth to the "bandwidth discovery" problem. We propose a novel solution to this problem, in which if the network appears quiescent we seize the available bandwidth all at once. This approach has a number of hazards associated with it, and our experiment methodology includes ways of assessing these hazards as well as the general effectiveness of the discovery algorithm.

Our final experiment derives from the observation that as network speeds increase, information derived during the current connection becomes harder and harder to use because connections become increasingly short-lived. We therefore propose investigating the degree to which information learned from previous connections can be incorporated into new connections, using the same framework as in the previous two experiments.

In summary, we believe the development of sound information-gathering techniques will make a vital contribution to the successful operation of tomorrow's Internet.

10 Acknowledgments

This proposal benefited greatly from discussions with Domenico Ferrari, Sally Floyd, Van Jacobson, and Steve McCanne. In particular, the illuminating metaphor of acks as "RADAR echoes", the general argument that endpoints can discover a great deal about the network's state, and the pressing need for devising ways for network connections to adapt based on the performance of past connections, all came from discussions with Van.

I would also like to thank the members of my qualifying committee, Tom Anderson, John Rice, and Pravin Varaiya, for their helpful comments and encouragement.

References

[BD92] Y. Baguette and A. Danthine, "Comparison of TP4, TCP and XTP, Part 2: Data Transfer Mechanisms", *European Transactions on Telecommunications and related technologies*, 3(6), pp. 625-635, Nov-Dec 1992.

[BMK88] D. Boggs, J. Mogul, and C. Kent, "Measured Capacity of an Ethernet: Myths and Reality", *Proceedings of SIGCOMM '88*, pp. 222-234, August 1988.

[Bo93] J-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet", *Proceedings of SIGCOMM '93*, pp. 289-298, September 1993.

[BOP94] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", *Proceedings of SIGCOMM '94*, pp. 24-35, September 1994.

[CW89] D. Cheriton and C. Williamson, "VMTP as the Transport Layer for High-Performance Distributed Systems", *IEEE Communications*, pp. 37-44, June 1989.

[CPB93a] K. Claffy, G. Polyzos, and H-W. Braun, "Traffic Characteristics of the T1 NSFNET Backbone", *Proceedings of INFOCOM '93*, San Francisco, March, 1993.

[CPB93b] K. Claffy, G. Polyzos, and H-W. Braun, "Measurement Considerations for Assessing Unidirectional Latencies", *Internetworking: Research and Experience*, 4(3), September 1993.

[CBP94] K. Claffy, H-W. Braun and G. Polyzos, "Tracking Long-Term Growth of the NSFNET", *Communications of the ACM*, 37(8), pp. 34-45, August 1994.

[CJRS89] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead", *IEEE Communications*, pp. 23-29, June 1989.

[CLZ87] D. Clark, M. Lambert, and L. Zhang, "NETBLT: A High Throughput Transport Protocol", *Proceedings of SIGCOMM '87*, pp. 353-359, 1987.

[Cl88] D. Clark, "The Design Philosophy of the DARPA Internet Protocols", *Proceedings of SIGCOMM '88*, pp. 106-114, August 1988.

[DJCME92] P. Danzig, S. Jamin, R. Cáceres, D. Mitzel, and D. Estrin, "An Empirical Workload Model for Driving Wide-area TCP/IP Network Simulations", *Internetworking: Research and Experience*, 3(1), pp. 1-26, 1992.

[DKS90] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", *Internetworking: Research and Experience*, 1(1), pp. 3-26, September 1990.

[DDKMRW90] W. Doeringer et al., "A Survey of Light-Weight Transport Protocols for High-Speed Networks", *IEEE Transactions on Communications*, 38(11), pp. 2025-2039, November 1990.

[FJ93] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, 1(4), pp. 397-413, August 1993.

[FJ94] S. Floyd and V. Jacobson, "The Synchronization of Periodic Routing Messages," *IEEE/ACM Transactions on Networking*, 2(2), pp. 122-136, April 1994.

[FL91] H. Fowler and W. Leland, "Local Area Network Traffic Characteristics, with Implications for Broadband Network Congestion Management", *IEEE JSAC*, 9(7), pp. 1139-1149, September 1991.

[GK80] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey", *IEEE Transactions on Communications*, 28(4), pp. 553-574, April 1980.

[He90] S. Heimlich, "Traffic Characterization of the NSFNET National Backbone", Proceedings of the 1990 Winter USENIX Conference, Washington, D.C.

[Ja88] V. Jacobson, "Congestion Avoidance and Control", *Proceedings of SIGCOMM '88*, pp. 314-329, August 1988.

[Ja81] J. Jaffe, "Bottleneck Flow Control", *IEEE Transactions on Communications*, 29(7), pp. 954-962, July 1981.

[Ja89] R. Jain, "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks", *Computer Communication Review*, 19(5), pp. 56-71, October 1989.

[Ja90a] R. Jain, "Myths About Congestion Management in High-Speed Networks", DEC-TR-726, Digital Equipment Corporation, October 1990.

[Ja90b] R. Jain, "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT", *Proceedings of SIGCOMM '90*, pp. 264-275, September 1990.

[Ke91] S. Keshav, "A Control-Theoretic Approach to Flow Control", *Proceedings of SIGCOMM '91*, pp. 3-15, September 1991.

[Kl76] L. Kleinrock, "Queueing Systems, Volume II: Computer Applications", John Wiley & Sons, 1976.

[LTWW94] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", *IEEE/ACM Transactions on Networking*, 2(1), pp. 1-15, February 1994.

[Mi83] D. Mills, "Internet Delay Experiments", RFC 889, Network Information Center, SRI International, Menlo Park, CA, 1983.

[MS90] D. Mitra and J. Seery, "Dynamic Adaptive Windows for High Speed Data Networks: Theory and Simulations (Extended Abstract)", *Proceedings of SIGCOMM '90*, pp. 30-37, September 1990.

[Mo92] J. Mogul, "Observing TCP Dynamics in Real Networks", *Proceedings of SIGCOMM '92*, pp. 305-317, August 1992.

[Na84] J. Nagle, "Congestion Control in IP/TCP Internetworks", RFC 896, Network Information Center, SRI International, Menlo Park, CA, 1984.

[NRS90] A. Netravali, W. Roome, K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol", *IEEE Transactions on Communications*, 38(11), pp. 2010-2024, November 1990.

[Pa94a] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections", to appear in *IEEE/ACM Transactions on Networking*, 2(4), August 1994.

[Pa94b] V. Paxson, "Growth Trends in Wide-Area TCP Connections", *IEEE Network*, 8(4), pp. 8-17, July/August 1994.

[PF94] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", *Proceedings of SIGCOMM '94*, pp. 257-268, September 1994.

[PS93] T. La Porta and M. Schwartz, "The MultiStream Protocol: A Highly Flexible High-Speed Transport Protocol", *IEEE JSAC*, 11(4), pp. 519-530, May 1993.

[RJ90] K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Transactions on Computer Systems*, 8(2), pp. 158-181, May 1990.

[SA91] D. Sanghi and A. Agrawala, "DTP: An Efficient Transport Protocol", UMIACS-TR-91-133, CS-TR 2767, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, October 1991.

[St94] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994.

[WC91] Z. Wang and J. Crowcroft, "A New Congestion Control Scheme: Slow Start and Search (Tri-S)", *Computer Communication Review*, 21(1), pp. 32-43, January 1991.

[WC92] Z. Wang and J. Crowcroft, "Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm", *Computer Communication Review*, 22(2), pp. 9-16, April 1992.

[WM87] R. Watson and S. Mamrak, "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices", *ACM Transactions on Computer Systems*, 5(2), pp. 97-120, May 1987.

[WS94] G. Wright and W. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, forthcoming.

[ZSC91] L. Zhang, S. Shenker, and D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", *Proceedings of SIGCOMM '91*, pp. 133-147, September 1991.