ThunderDome: Discovering Upload Constraints Using Decentralized Bandwidth Tournaments

John R. Douceur[1], James W. Mickens[1], Thomas Moscibroda[1], Debmalya Panigrahi[2]
[1] Microsoft Research, Redmond, WA, {johndo, mickens, moscitho}@microsoft.com
[2] Massachusetts Institute of Technology (MIT), Cambridge, MA, debmalya@mit.edu

ABSTRACT

ThunderDome is a system for collaboratively measuring upload bandwidths in ad-hoc peer-to-peer systems. It works by scheduling *bandwidth probes* between pairs of hosts, wherein each pairwise exchange reveals the upload constraint of one participant. Using the abstraction of *bandwidth tournaments*, unresolved hosts are successively paired with each other until every peer knows its upload bandwidth. To recover from measurement errors that corrupt its tournament schedule, ThunderDome aggregates multiple probe results for each host, avoiding pathological bandwidth estimations that would otherwise occur in systems with heterogeneous bandwidth distributions. For scalability, the coordination of probes is distributed across the hosts. Simulations on empirical and analytic bandwidth distributions—validated with wide-area PlanetLab experiments—show that ThunderDome efficiently yields upload bandwidth estimates that are robust to measurement error.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Algorithms, Experimentation, Measurement

1. INTRODUCTION

In peer-to-peer systems, upload constraints are often the key determinant of scalability and performance. Thus, P2P systems often try to optimize their usage of the participants' upload bandwidth. For example, the Donnybrook gaming framework [1] uses peers with high upload bandwidths to distribute player updates to poorly connected hosts. More generally, multicast systems [2, 3] and media streaming services [9, 15] can increase delivery rates by buildling dissemination topologies with well-provisioned hosts at the core. In many of these systems, the most powerful optimizations require more than a ranking of peers by their upload bandwidths; instead, systems like Donnybrook assign specific tasks to peers based on the exact upload bandwidth available to each peer. Since available bandwidth is dynamic, it must be estimated on demand, at the time of use.

All of these systems assume that the upload constraints are known *a priori*. However, actually determining these constraints is an open research problem. A host can use a wide variety of tools [10, 13, 17, 19, 20, 21, 25] to measure the one-way available bandwidth between itself and another peer. However, this measurement does not reveal the sender's maximum upload speed—instead, it reveals the minimum of the sender's upload constraint and the receiver's download constraint. Disambiguating the two is very important for constructing bandwidth-aware topologies. For example, suppose that the one-way transfer rate between two peers is 30 Kbps. At least one of the peers has a slow connection, *but one of the peers may have a very fast connection*. We want to identify and leverage the latter type of host, but we cannot do so by naively applying one-way bandwidth estimators.

If the sender could definitively identify a remote endpoint with a greater download capacity that its (currently undetermined) upload bandwidth, the sender could trivially determine its upload constraint. Unfortunately, in an ad-hoc P2P system, hosts usually lack *a priori* knowledge of such well-provisioned endpoints. Even if high-bandwidth sinks are well-known, their utility is diminished if too many senders initiate concurrent transfers and reduce the available per-sender download bandwidth.

To address these challenges, we introduce ThunderDome, a system for collaboratively measuring available upload bandwidths in ad-hoc peer-to-peer environments. ThunderDome's measurement primitive is a *pairwise bandwidth probe* which measures the transfer rate between two hosts in both directions. Given the asymmetric upload/download speeds of popular access technologies like DSL, the slower of the two unidirectional transfers reveals the upload bandwidth of the sending node. Using *bandwidth tournaments*, ThunderDome arranges additional bandwidth probes for unresolved nodes. An unresolved peer is successively paired with other such hosts until it encounters a peer whose download bandwidth is high enough to reveal its upload bandwidth. For scalability, the coordination of the probing activity is distributed among the peers themselves and requires minimal centralized intervention. Analysis shows that the running time of ThunderDome is logarithmic in the system size.

In practice, unidirectional bandwidth probes are subject to measurement errors [11, 17, 23, 26]. These errors can interfere with the determination of which host was resolved in a pairwise bandwidth probe. To prevent these errors from cascading throughout the tournament, ThunderDome uses additional "tightening" probes that do not add to the run-

CoNEXT'09, December 1–4, 2009, Rome, Italy.

ning time of the algorithm. ThunderDome then uses statistical techniques to aggregate the probe results for a given host and eliminate outliers.

Since ThunderDome relies on active network probing, a natural concern is whether a real deployment would generate excessive traffic. Fortunately, a ThunderDome host is involved in at most one transfer at any time, and modern one-way bandwidth estimators do not produce an overwhelming amount of taffic. For example, the Spruce tool [25] generates 300 KB of traffic per measurement, and IGI [10] produces 130 KB. Given that an average web page is 300 KB [27], this volume of probing traffic is quite reasonable.

We evaluate ThunderDome with extensive simulations on a real, empirically measured bandwidth distribution, as well as an analytic distribution that allows us to parametrically consider a wide range of alternative network environments. Our experiments show that, in a small number of tournament rounds, ThunderDome produces upload bandwidth estimates with accuracy comparable to the inherent measurement error in the network. We validate our simulations with wide-area experiments using PlanetLab [24].

In summary, we make the following contributions:

- We describe how bandwidth tournaments can determine upload constraints in ad-hoc P2P systems in an efficient and scalable manner.
- We identify problems arising from measurement errors, and propose statistical techniques to mitigate the effects of such errors.
- We propose a fully distributed implementation for bandwidth tournaments.
- We validate the accuracy of ThunderDome by means of extensive simulations using empirical and analytic bandwidth distributions. We also describe an experimental validation on PlanetLab.

2. BACKGROUND AND RELATED WORK

An Internet path connecting two hosts consists of multiple routers and physical links. The *capacity* of an individual link is the maximum rate at which it can transmit packets. At any given time, the link's *available bandwidth* is its unused residual capacity. The capacity of an end-to-end path is the minimum capacity among its constituent links. The path's available bandwidth is defined in a similar way. In this paper, we focus on determining available bandwidths in ad-hoc distributed groups. We defer a discussion of capacity estimation to other work [5, 7, 16].

There are many tools for estimating the unidirectional available bandwidth along an Internet path [10, 11, 13, 19, 21, 20, 25, 17]. At a high-level, these tools use one of two techniques. *Packet rate* tools [10, 13, 19, 21] gradually increase the packet generation rate at the sender, defining the available bandwidth as the highest send rate that experiences no losses at the receiver. *Packet gap* tools [10, 20, 25] issue pairs of packets with carefully chosen inter-packet spacings. This gap will increase as each pair traverses the path and the trailing packet is queued behind cross-traffic that arrives after the lead packet hits each router. Using mathematical models, the receiver can determine the volume of this cross-traffic and the remaining available bandwidth. ThunderDome's pairwise bandwidth probes require some way of determining unidirectional transfer speeds, but ThunderDome is agnostic as to which tool is used.

BRoute [11] minimizes probing traffic by exploiting two observations: most bottleneck links reside at the edge of the network, and in large systems, each edge link is likely to be shared by multiple end-to-end paths. By only probing edge links, BRoute can identify most sources of congestion without exhaustive introspection of each link in a path. Furthermore, the probing cost for each edge segment is amortized across the set of peers which use that segment.

Unlike the other tools described above, BRoute uses dedicated network infrastructure—hosts discover their edge links by issuing traceroutes to well-known landmark nodes. BRoute also requires BGP data to determine which edge segments connect end hosts. In our problem domain, groups are ad-hoc, meaning that dedicated infrastructure is unlikely to exist. Our groups are also composed of "regular" end-users who lack access to privileged BGP feeds.

Like all measurement tools, one-way bandwidth estimators do not generate perfectly accurate results. Some of the measurement error arises from the simplifying assumptions that these tools make; for example, tools often assume that all routers use a FIFO queuing discipline, and that the volume of cross-traffic is stationary. Resource constraints on the measurement hosts can also cause poor estimations. In particular, many tools require fine-grained timers to control the rate at which packets are sent. They also require accurate timestamps for received packets. On highly loaded machines, the OS may be unable to provide the necessary fidelity [24]. Even a lightly loaded machine may underestimate a high bandwidth link if the native timestamp resolution is too coarse [23].

Because of these issues, the output of a bandwidth estimation tool should not be taken as ground truth. For example, pathchirp [21] underestimates available bandwidth when the volume of cross-traffic is low, but overestimates when cross-traffic is high [26]. Bandwidth estimates are also sensitive to the size of probe packets [10, 17, 21, 23], but different paths may be optimally measured with different packet sizes, and generating this mapping is not straightforward. Thus, measurement jitter is often non-trivial—in controlled settings, the standard deviation of measurement can be 30% or more [11, 17, 23, 26]. In Sections 4 and 6, we show how measurement noise can lead to dramatic underestimates of upload bandwidth.

3. ALGORITHMIC FOUNDATIONS

In this section, we formally define the bandwidth estimation problem in P2P networks and present basic algorithms to solve it. Throughout this section, we use an abstract model that ignores several important issues like measurement error. However, the practical algorithms that we describe later are derived from these basic techniques.

3.1 Participant Model

ThunderDome is designed for online gaming or collaborative streaming applications in which users gather to participate in a service and then use the service for tens of minutes or longer. At a minimum, ThunderDome runs at initialization time, providing the service with peer upload constraints that can guide the construction of bandwidth-aware communication topologies. Subsets of the peers may run the ThunderDome protocol later if they believe that their available upload bandwidths have changed.

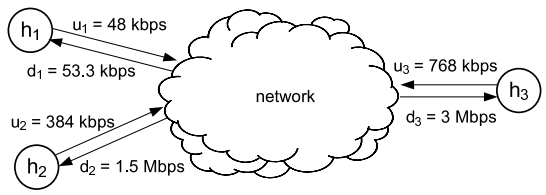


Figure 1: Bandwidth probe $h_1 \leftrightarrow h_3$ reveals only h_1's upload bandwidth, since $P(h_3 \rightarrow h_1) = d_1$. In contrast, $h_2 \leftrightarrow h_3$ reveals both uploads (i.e., $P(h_2 \rightarrow h_3) = u_2$ and $P(h_3 \rightarrow h_2) = u_3$), but h_3 cannot be sure that the measurement $P(h_3 \rightarrow h_2)$ is indeed u_3.

ThunderDome assumes that hosts accurately report the results of bandwidth probes and do not try to attack the distributed measurement protocol. It also assumes that each host has a public IP address, or resides behind a cone NAT [22] that allows outside parties to initiate communication with the host. If a peer relies on a relay node to communicate with the outside world, the relay node acts as the peer's representative in the ThunderDome protocol.

3.2 Network Model & Problem Definition

Our network model is based on two primary assumptions.

- *Hub-and-Spoke Model:* We assume that bandwidth bottlenecks occur on last mile edge links, and that the core of the Internet has essentially unlimited bandwidth. Although "middle mile" congestion between adjacent ISPs can sometimes induce packet losses [18], the Internet core is generally much better provisioned than the edge. In Section 6.2, we validate this assumption for target transfer bandwidths of up to 768 Kbps.
- *Directional Asymmetry:* We assume that for each host h_i, h_i's download bandwidth is greater than or equal to its upload bandwidth. This asymmetry is inherent for the vast majority of network access technologies.

For now, we assume that there are no measurement errors, i.e., we can determine one-way path bandwidths with perfect accuracy. We will relax this assumption in Section 4. For now, our goal is to develop basic algorithms that allow every node to quickly determine its upload speed. *Bandwidth probes* are the building blocks for these algorithms.

Bandwidth Probe: A bandwidth probe is a pairing of two hosts h_i and h_j such that h_i transmits data to h_j at full speed and then vice versa. We denote such a bandwidth probe as $h_i \leftrightarrow h_j$. The result of a bandwidth probe $h_i \leftrightarrow h_j$ is two measurements, denoted by $P(h_i \rightarrow h_j)$ and $P(h_j \rightarrow h_i)$. $P(h_i \rightarrow h_j)$ is the speed at which data was transmitted from h_i to h_j. This is the minimum of h_i's upload and h_j's download, i.e.,

$$P(h_i \rightarrow h_j) = \min\{u_i, d_j\}.$$

Similarly, the transfer speed in the opposite direction reveals the minimum of h_j's upload and h_i's download, i.e.,

$$P(h_j \rightarrow h_i) = \min\{u_j, d_i\}.$$

For each bandwidth probe, one of the two hosts is the *winner*, denoted by $W(h_i \leftrightarrow h_j)$. The other peer is the *loser* $L(h_i \leftrightarrow h_j)$. Specifically, if $P(h_i \rightarrow h_j) > P(h_j \rightarrow h_i)$, then h_i is the winner and h_j is the loser. If $P(h_j \rightarrow h_i) > P(h_i \rightarrow h_j)$, then $W(h_i \leftrightarrow h_j) = h_j$ and $L(h_i \leftrightarrow h_j) = h_i$ (see Figure 1).

Given $P(h_i \rightarrow h_j)$ and $P(h_j \rightarrow h_i)$, and our assumption that $d_i \geq u_i$ and $d_j \geq u_j$, we can derive the following information from a pairwise exchange $h_i \leftrightarrow h_j$:

- If $P(h_i \rightarrow h_j) \geq P(h_j \rightarrow h_i)$, it follows that $u_j = P(h_j \rightarrow h_i)$. If $P(h_i \rightarrow h_j) \leq P(h_j \rightarrow h_i)$ then $u_i = P(h_i \rightarrow h_j)$. In other words, in the absence of measurement error, a pairwise bandwidth probe reveals (at least) the *upload bandwidth of the loser*.
- While the smaller of the two directional probes corresponds to the upload bandwidth of the loser, the larger probe can either be the loser's download bandwidth *or* the winner's upload bandwidth.

Disambiguating the latter condition is difficult. For example, if $u_i < u_j$, then a bandwidth probe will reveal that $u_i = P(h_i \rightarrow h_j)$. If we also knew that $d_i \geq u_j$, the probe could additonally reveal $u_j = P(h_j \rightarrow h_i)$. Unfortunately, there is no *a priori* way for the hosts to determine whether $d_i \geq u_j$ is true, i.e., whether the bottleneck in the transfer from h_j to h_i is h_j's upload speed or h_i's download speed (see Figure 1). This missing information is the primary motivation for the use of bandwidth tournaments to resolve each host's upload constraint. By employing a series of bandwidth probes, we can eliminate this uncertainty.

With these definitions, we can now formally define the bandwidth estimation problem in P2P networks.

Bandwidth Estimation Problem: *Let H denote the set of n participating hosts $H = \{h_1, \ldots, h_n\}$. Each host is connected to the Internet via an access link with download bandwidth d_i and upload bandwidth u_i, such that $d_i \geq u_i$ for all i. Let $h_{max\_u_i}$ be the host with the highest upload bandwidth. Initially, no host knows its upload or download bandwidth. The goal is to employ a series of bandwidth probes such that in the end, all nodes but $h_{max\_u_i}$ know their u_i.*[1]

We assume that time is divided into *rounds*, where a round is the unit of time required to conduct one pairwise bandwidth probe. In each round, every host can participate in one pairwise probe with another peer; thus, up to $n/2$ exchanges can be done in parallel. When comparing various algorithms for solving the bandwidth estimation problem, the key evaluation metric is running time.

Time Complexity: The time complexity $TC(A)$ of an algorithm A is the maximum number of rounds during which there is at least one bandwidth probe.

3.3 Simple Approaches

Centralized Server (CS): In small-scale systems, a common way to determine upload speeds is to have nodes send a bandwidth probe to a central server that is known to have very high download bandwidth. This approach does not exploit the upstream and downstream bandwidths of the other hosts in the system, but it has the benefit of being simple and easy to implement. Unfortunately, as n gets larger, the time complexity of such an approach scales poorly. Specifically, given a server with download speed d_s, the time complexity

[1] $h_{max\_u_i}$ will never lose a pairwise exchange, so it can never definitively determine its upload bandwidth. However, it can determine a lower bound for this bandwidth as the highest upload speed that it observed across all pairwise exchanges. This lower bound will be an exact estimate if there exists d_j greater than $h_{max\_u_i}$'s upload speed, and $h_{max\_u_i}$ is ever paired with h_j.

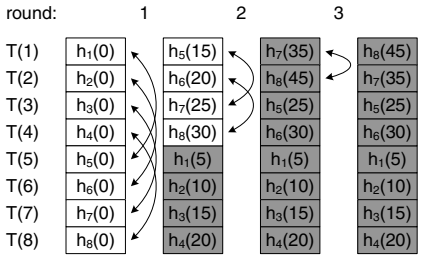


Figure 2: Example progression of basic algorithm ($n = 8$, $u_i = 5 * i$, $d_i = u_i + 10$). Estimates are in parentheses, and are gray when correct. After round 2, h_7 and h_8 are correct but not yet known to be so.

is at least $T(CS) \geq \sum_{h_i \in H} u_i/d_s$, i.e., even with optimal coordination, the time complexity grows linearly.

Random Sampling (RS): A more clever idea is to increase parallelism by making use of the available download bandwidth of *all* participating hosts, not just that of the server. Indeed, if every hosts' download bandwidth was higher than every host's upload bandwidth ($d_i \geq u_j, \forall i, j$), the problem would be trivial in the absence of measurement errors. After a single round of probing with random host pairs, every host would know its upload bandwidth. In practice, of course, some downloads will be lower than some of the uploads, rendering the above scheme unusable.

A multi-round peer-to-peer approach would partially alleviate the problem. In each round, every host is randomly paired with another host, resulting in $n/2$ independent parallel bandwidth probes in each round. Once a host loses a pairwise exchange, it knows its upload bandwidth.

Assume hosts $h_1, \ldots, h_n$ are ordered in non-decreasing order of their upload bandwidth $u_1 \leq u_2 \leq, \ldots, \leq u_n$. In worst-case bandwidth distributions, the random sampling approach performs poorly.

THEOREM 3.1. *There are bandwidth distributions for which random sampling achieves an expected time complexity of* $TC(RS) = n/2$.

PROOF. If $u_n > d_i$ for all $i = 1, \ldots n-2$, host h_n can only determine its upload bandwidth if it is paired with host h_{n-1} which, in expectation, takes time $n/2$. □

3.4 ThunderDome: Bandwidth Tournaments

We now present our basic algorithm, which achieves better guarantees than the simple approaches discussed in the previous section. In the absence of measurement errors, this algorithm has time complexity of $O(\log n)$ for any bandwidth distribution.

In the absence of measurement errors, the idea of ThunderDome is very simple. Every bandwidth probe reveals the upload speed of the loser, so only the winners need to continue being paired with each other. As described in Algorithm 1, ThunderDome pairs winners with each other. As soon as a node loses a bandwidth probe, it no longer participates in the subsequent rounds. Whereas random sampling had bad worst-case performance, it is easy to see that ThunderDome has a time complexity of $\log n$ regardless of the bandwidth distribution, because in every round, at least half of the remaining hosts resolve their upload bandwidth. Figure 2 demonstrates the operation of the algorithm.

Algorithm 1 Basic ThunderDome Algorithm

Input: Set of n hosts $\{h_i | i \in [1..n]\}$
  Count k of rounds to run
Output: None; hosts accumulate data as a side effect

```
Set T(i) := h_i for each i ∈ [1..n]
for r := 1 to k do
  for all i ∈ [1..n/2^r] do
    Set j := n/2^r + i
    Perform probes T(i) ↔ T(j)
    if host T(j) wins then
      Swap T(i) and T(j)
    end if
  end for
end for
```

THEOREM 3.2. *In the absence of measurement errors, it holds for every bandwidth distribution that after k rounds, at least* $n/2^k$ *hosts know their upload bandwidth. Therefore,* $TC(TD) = \log n$.

Given this theorem, we see that with respect to random sampling, ThunderDome provides an exponentially improved time complexity for worst-case bandwidth distributions (from $\Omega(n)$ to $O(\log n)$).

4. DEALING WITH PROBING ERRORS

Up to this point, we have assumed that bandwidth probes yield accurate results. However, bandwidth estimation tools can produce significantly erroneous results, with individual measurements deviating from true bandwidths by 30% or more [11, 17, 23, 26]. These errors arise because the behavior of the routing core is largely opaque to end hosts, and behavioral assumptions made by end-host tools are often violated (§2). The resulting errors can alter the control flow of bandwidth-aware systems and degrade performance.

For reference, we define the terms:

- *fractional probing error:* $\frac{BW_{probe}}{BW_{true}} - 1.0$
- *fractional estimation error:* $\frac{BW_{estimate}}{BW_{true}} - 1.0$

Since a bandwidth estimate is merely a result from one of two bandwidth probes, one might think that the estimation error will not exceed the probing error. Surprisingly, this is not correct. As we explain below, the estimation error may be as high as $\max_{i \in [1..n]} u_i - \min_{i \in [1..n]} d_i$, i.e., the difference between the largest upload speed and the smallest download speed of any host in the system. Depending on the distribution of host bandwidths, this spread may grow arbitrarily large with respect to the probing error. In particular, the basic ThunderDome algorithm can dramatically *underestimate* upload bandwidth. Thus, we must modify the algorithm to hedge against faulty probe measurements.

4.1 Problem: Mismeasurement Occlusion

Recall that one of ThunderDome's enabling assumptions is directional asymmetry: for each host h_i, $u_i \leq d_i$. However, the difference between u_i and d_i may be smaller than the probing error. This can cause ThunderDome to reach an incorrect conclusion about which host's upload bandwidth is determined by a particular probe.

Consider the system of Figure 1 and a probe between h_1 and h_3. In the absence of probing error, $P(h_1 \rightarrow h_3) = 48$ kbps and $P(h_3 \rightarrow h_1) = 53.3$ kbps. Since $P(h_1 \rightarrow h_3) <$

$P(h_3 \rightarrow h_1)$, ThunderDome will correctly conclude that 48 kbps is h_1's upload bandwidth.

Now consider a fractional probing error of up to 15%. In the worst case, $P(h_1 \rightarrow h_3) = 55.2$ kbps and $P(h_3 \rightarrow h_1) = 45.3$ kbps. Since $P(h_3 \rightarrow h_1) < P(h_1 \rightarrow h_3)$, ThunderDome will wrongly conclude that 45.3 kbps is h_3's upload bandwidth. Since the true value of u_3 is 768 kbps, the magnitude of the fractional estimation error is 94%, even though the probing error is only 15%.

We refer to this phenomenon as *mismeasurement occlusion*: a host's upload bandwidth is occluded by an incorrect measurement of its probe-partner's download bandwidth. As probing error increases, a greater fraction of hosts will be vulnerable to such occlusions. The magnitude of the estimation error depends on the bandwidth skew in the network. Let $\hat{u}_i$ and $\hat{d}_i$ represent the (possibly erroneous) measurement of h_i's bandwidth in a pairwise exchange. In the worst case, the estimation error for any other host h_j is $\min_{i \in [1..j-1, j+1..n]}\{\hat{d}_i | \hat{u}_i > \hat{d}_i\} - u_j$. As bandwidth heterogeneity grows, this worst-case estimation error increases, unbounded by probing error.

In practice, underestimation bias will occur in systems with a mix of high-speed connections and either wireless devices or dial-ups. Survey data indicates that at least 30% of Americans connect to the Internet using dial-up modems [6], whose upload speeds are an order of magnitude smaller than that of mid-range DSL and cable connections.

4.2 Tightening

In the basic ThunderDome algorithm, once a host loses a bandwidth probe, its bandwidth estimate is established and never revised. To combat underestimation bias resulting from mismeasurement occlusion, we can allow losers to perform additional probes with other hosts. Each losing probe $P(h_i \rightarrow h_j)$ provides a *provisional lower bound* on h_i's true upload bandwidth. Given a set of such bounds from multiple probes from h_i, we can use the highest one as our estimate of u_i. Thus, a high-bandwidth node with a dramatically underestimated upload speed (due to mismeasurement occlusion) can revise its estimate upward it if subsequently pairs with another high-speed node.

A straightforward way to provide such additional pairings is to perform one or more *tightening rounds* at the end. In each tightening round, we first sort all hosts by their upload estimates: $\forall i \in [1..n-1] : \hat{u}_{s(i)} < \hat{u}_{s(i+1)}$. Then, we perform probes between pairs of hosts with widely separated ranks: $\forall i \in [1..n/2] : h_{s(i)} \leftrightarrow h_{s(i+n/2)}$. Each tightening round requires $n/2$ probes, which can all proceed in parallel.

4.3 Inline Tightening

Adding tightening rounds at the end has two disadvantages. First, it increases the running time of the algorithm. Second, it treats all hosts identically, even though some hosts are more likely to experience mismeasurement occlusion than others.

To elaborate on the second point, note that a host that wins a few rounds will tend to have higher measured bandwidth than a host that loses earlier in the tournament. Furthermore, this winning host will subsequently be paired with other winning hosts, which also have high measured bandwidths. Consequently, the potential for mismeasurement occlusion decreases as the tournament proceeds. Thus, the hosts most in need of tightening are the losers of early rounds.

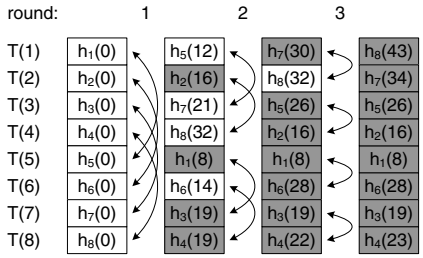


Figure 3: Example progression of algorithm with inline tightening, for $n = 8$, $u_i = 5 * i$, $d_i = u_i + 10$, and error ± 6. In round 1, h_6 incorrectly loses to h_2. In round 2, a probe with h_4 corrects h_6's underestimate.

We can address both of these issues by *inlining* the tightening procedure. When a host loses round r, rather than merely waiting around for $k - r$ more rounds and then performing tightening, the host can perform its tightening steps during these $k - r$ rounds, without adding to the overall running time. Furthermore, hosts that lose early—and which thus have the most need for tightening—have more time to perform a greater number of tightening rounds.

Algorithm 2 shows how ThunderDome performs this *inline tightening*, and Figure 3 gives an example. Essentially, nodes are placed at the vertexes of a $(\log n)$-dimensional hypercube. In each round, probes are exchanged between hosts that are paired along one axis of the hypercube.

α-tightening: An advantage of the inline tightening approach is that the most vulnerable hosts perform many rounds of tightening. However, now that each host obtains multiple bandwidth estimates during the tournament, ThunderDome must define a way for a node to aggregate these values and derive a final estimate. Taking the *maximum* over all samples can lead to overestimation, which follows directly from the theory of order statistics. Conversely, if we take the *mean* over all samples, measurement errors will be averaged out, but the mean will incorporate even very low estimates caused by mismeasurement occlusion, undermining the very purpose of tightening.

To obtain the best of both schemes, we introduce a technique we call *α-tightening*. If the host's largest measurement sample is $\hat{u}_{max}$, then we compute the host's bandwidth estimate as the mean over all samples in the range $[\alpha \cdot \hat{u}_{max}, \hat{u}_{max}]$ for some specified value of α.[2] This preserves the averaging properties of the mean, without incorporating erroneously low estimates.

5. DISTRIBUTED THUNDERDOME

In ThunderDome, the probing effort is distributed among the hosts, which execute their bandwidth probes in parallel during each round. However, as presented above, the hosts require coordination after each round to determine the pairings for the subsequent round. The straightforward way to coordinate the hosts is with a central *leader*. In each round r, the leader receives all probe results from round $r - 1$, computes a schedule for the hosts according to one pass of the outer loop in Algorithm 1 or 2, and tells each host who to pair with in the next round.

[2]Note that for $\alpha = 1$ and $\alpha = 0$, α-tightening reduces to the maximum and mean, respectively.

Algorithm 2 ThunderDome with Inline Tightening

Input: Set of n hosts $\{h_i | i \in [1..n]\}$, number of rounds k
Output: None; hosts accumulate data as a side effect

```
Set T(i) := h_i for each i ∈ [1..n]
for r := 1 to k do
  for all x ∈ [0..2^(r-1) − 1] do
    for all y ∈ [0..n/2^r − 1] do
      Set i := n/2^(r-1) x + y + 1
      Set j := i + n/2^r
      Perform probes between hosts T(i) and T(j)
      if host T(j) wins then
        Swap T(i) and T(j)
      end if
    end for
  end for
end for
```

In large systems, the network load on the leader can become substantial. Indeed, more time could be spent by the leader (serially) sending instructions to each host than by the hosts (in parallel) sending probes to each other. In this section, we discuss two ways to deal with this problem: partitioning and distributed coordination.

5.1 Partitioning

The simplest way to reduce the communication load on the leader is to partition the n hosts into g subsets, nominate one host in each subset to serve as a sub-leader for that subset, and perform g instances of the algorithm in parallel, with n/g hosts in each instance. In some cases, this might work acceptably, but there are two reasons that it may be insufficient.

First, as our evaluation (§6) shows, ThunderDome's estimation accuracy is related to the number k of probing rounds. In smaller systems, $k \leq \log n$, so partitioned subsets may be unable to perform enough rounds to reach the desired accuracy.

Second, recall that our problem definition (§3, footnote 1) allows the highest upload bandwidth to remain undetermined if the download bandwidths of all other hosts in the set are lower than this highest upload bandwidth. We can evaluate the probability that this occurs as a function of the number of hosts. We characterize host bandwidths using a joint probability density function b with respect to upload and download bandwidths governed by random variables U and D: $b(u,d) = P[U = u, D = d]$.

We derive the failure probability f for a host in a tournament among n hosts, whose bandwidths follow joint density function b. Let V_n be the n'th order statistic of U. That is, random variable V_n governs the distribution of the highest upload bandwidth among the n hosts. A tournament will fail for one in n hosts if (1) the host with highest upload bandwidth has upload bandwidth x and (2) the $n-1$ hosts with lower upload bandwidth have download bandwidths that are less than x:

$$f(b,n) = \frac{1}{n} \int_0^\infty P[V_n = x]\, P[D \leq x \,|\, U \leq x]^{n-1}\, dx \quad (1)$$

From basic order statistics:

$$P[V_n = x] \;=\; n\, P[U = x]\, P[U \leq x]^{n-1} \quad (2)$$

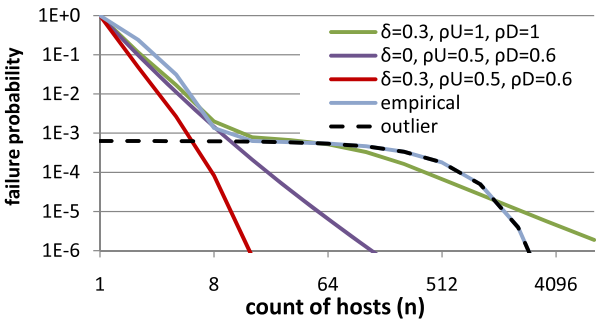


Figure 4: Probability of failing to accurately determine a hosts's upload speed by a tournament among n hosts, for various bandwidth distributions. One outlier dominates the empirical curve for $n > 16$.

From the definition of conditional probability:

$$P[D \leq x \,|\, U \leq x] = \frac{P[D \leq x \wedge U \leq x]}{P[U \leq x]} \quad (3)$$

The property of directional asymmetry implies that:

$$P[D \leq x \wedge U \leq x] = P[D \leq x] \quad (4)$$

A tournament will fail for a host whose upload bandwidth is higher than all other host's download bandwidths. Therefore, substituting Eq. 4 into Eq. 3, substituting Eqs. 2-3 into Eq. 1, and simplifying yields

$$f(b,n) = \int_0^\infty P[U = x]\, P[D \leq x]^{n-1}\, dx \quad (5)$$

Interestingly, this value depends only on the marginal distributions of the two random variables, not their joint distribution.[3]

Figure 4 plots f versus n for empirical and analytical bandwidth distributions (§6.1). For the analytical distribution, the effect on f is governed by three parameters: δ, $\rho_U = \sigma_U/(\mu_D - \mu_U)$, and $\rho_D = \sigma_D/(\mu_D - \mu_U)$. The values ($\delta = 0.3$, $\rho_U = 0.5$, $\rho_D = 0.6$) are derived from an analytical fit to the empirical distribution.

The failure curve for the empirical distribution does not well match the curve for the fitted analytical distribution. Instead, it is dominated by an outlier: a large bandwidth value in fraction p of the upload distribution, which is less than merely fraction q of the download distribution. This causes f to be dominated by $p\,(1-q)^{n-1}$. In our empirical distribution, $p = 0.0006$ and $q = 0.0025$.

This analysis shows that simple partitioning may have an excessively detrimental effect on the result of the bandwidth tournament. Even with hundreds of hosts, the failure probability is non-trivial. Moreover, this probability can be highly dependent on characteristics of a small fraction of the population, making it difficult to quantify.

5.2 Distributed Coordination

An alternative approach to reducing the leader's communication load is to distribute the coordination effort among all the peers while preserving the probing schedule that would be generated by centralized coordination. To do so, we employ a set of $n - n/2^k$ *coordinators*, each of which is responsible for coordinating a probe between a pair of hosts.

[3]The derivation employs the property of directional asymmetry, which the joint distribution is assumed to satisfy.

Algorithm 3 Distributed ThunderDome—Host

Input: Initial coordinator c_{init} for the host
Output: None; host accumulates data as a side effect

```
1: Set c := c_init
2: while c ≠ ⊥ do
3:    Ask coordinator c for instructions
4:    Receive partner h' and next coordinator c_next from c
5:    Perform probes to/from host h'
6:    if current host wins round then
7:       c := c_next
8:    else
9:       stop
10:   end if
11: end while
```

Algorithm 4 Distributed ThunderDome—Coordinator

Input: Parent coordinator c_{next}
Output: None

```
1: Wait for messages from any two hosts, h_i and h_j
2: Send (h_j and c_next) to h_i
3: Send (h_i and c_next) to h_j
4: stop
```

In the distributed version of the basic ThunderDome, each host is initially assigned to a coordinator c_{init}. In each round, the host contacts its coordinator, learns which host it should probe, executes its probe, and updates its coordinator based on the result of the probe (Algorithm 3).

The job of each coordinator is even simpler. It waits to be contacted by two hosts, at which point it tells the hosts about each other. It also tells the hosts about its *parent* coordinator c_{next}, which it received at initialization (see Algorithm 4). This key initialization step sets c_{init} for all hosts and c_{next} for all coordinators according to Algorithm 5. Figure 5 illustrates the resulting topology for $n = 8$.

The distributed algorithm results in the *exact same* probe schedule as the centralized algorithm. To state this more formally, we introduce two pieces of notation. First, the winner function $W(h, h')$ returns the host that wins a bandwidth probe between hosts h and h'. Second, for tournament algorithm A and for $r \in [1..k]$, the match function $M^A(r)$ returns a set of unordered pairs of hosts; $\{h, h'\} \in M^A(r)$ iff algorithm A pairs hosts h and h' in round r.

THEOREM 5.1. *For any power-of-two n, $k \leq \log n$, $r \in [1..k]$, and fixed W, it holds that $M^{centr}(r) = M^{distr}(r)$.*

The proof for this theorem is straightforward though rather tedious. It mainly shows that the numerical manipulations in Algorithm 5 are isomorphic to those in Algorithm 1.

Coordinators are merely hosts acting in a different role, but we designate them separately for clarity. A simple mapping between hosts and their coordinator roles is:

$$\forall r \in [1..k], i \in [1..n/2^r], j = n/2^r + i : \; c_i^r = h_j$$

5.3 Distributed Inline Tightening

The distributed version of ThunderDome with inline tightening is similar to the above. When the host updates its coordinator based on the result of a probe, it selects one of two next coordinators, c_{win} and c_{lose}, as specified in Algorithm 6. Similarly, each coordinator is assigned two parent coordinators, c_{win} and c_{lose}, which it tells the hosts about when contacted, as specified in Algorithm 7. Again, the crux is the initialization step, which sets values for c_{init}, c_{win}, and c_{lose} according to Algorithm 8. Figure 6 illustrates the resulting topology for $n = 8$. As in the case without tightening, the distributed algorithm results in the *exact same* set of probes as the centralized algorithm.

Algorithm 5 Distributed ThunderDome—Initialization

Input: Set of n hosts $\{h_i | i \in [1..n]\}$
    Count k of rounds to run
    Set of $n - n/2^k$ coordinators $\{c_i^r | r \in [1..k], i \in [1..n/2^r]\}$
Output: Assignment of c_{init} for each host
    Assignment of c_{next} for each coordinator

```
1: for all r ∈ [1..k] do
2:    for all i ∈ [1..n/2^r] do
3:       Set j := n/2^r + i
4:       if r = 1 then
5:          Set h_i.c_init := c_i^1
6:          Set h_j.c_init := c_i^1
7:       else
8:          Set c_i^{r-1}.c_next := c_i^r
9:          Set c_j^{r-1}.c_next := c_i^r
10:      end if
11:   end for
12: end for
13: for all i ∈ [1..n/2^k] do
14:    Set c_i^k.c_next :=⊥
15: end for
```

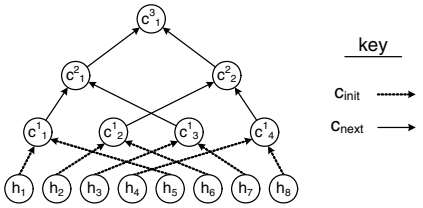


Figure 5: Topology of basic distributed algorithm for $n = 8$. Note the visual isomorphism to Figure 2.

THEOREM 5.2. *For any power-of-two n, $k \leq \log n$, $r \in [1..k]$, and fixed W, it holds that $M^{c-tight}(r) = M^{d-tight}(r)$.*

As with Theorem 5.1, the proof for this theorem is a straightforward though very tedious verification of the isomorphism between the numerical manipulations in Algorithm 8 and those in Algorithm 2.

In the absence of tightening, there are strictly fewer coordinators than hosts. This is not the case for the inline tightening algorithm, so it would be prudent to avoid assigning any host to multiple coordination roles that demand simultaneous effort. Since each coordinator functions for only one round, this is easily achieved:

$$\forall r \in [1..k], i \in [1..n/2], j = (r \bmod 2)n/2 + i : \; c_i^r = h_j$$

6. EVALUATION

In this section, we use a combination of simulations and Planetlab [24] experiments to evaluate ThunderDome's performance. First, we validate our hub-and-spoke network model, using Planetlab nodes to compare transfer speeds between WAN endpoints and LAN endpoints. To motivate the concept of tightening, we use simulations to demonstrate

Algorithm 6 Distr. Tighten ThunderDome—Host

Input: Initial coordinator c_{init} for the host
Output: None; host accumulates data as a side effect

Set $c := c_{init}$
while $c \neq \perp$ **do**
  Ask coordinator c for instructions
  Receive h', c_{win}, and c_{lose} from c
  Perform probes to/from host h'
  if current host wins round **then**
    $c := c_{win}$
  else
    $c := c_{lose}$
  end if
end while

Algorithm 7 Distr. Tighten ThunderDome—Coordinator

Input: Parent coordinators c_{win} and c_{lose}
Output: None

Wait for messages from any two hosts, h_i and h_j
Send (h_j, c_{win}, c_{lose}) to h_i
Send (h_i, c_{win}, c_{lose}) to h_j
stop

how measurement error can introduce non-linear underestimates of upload bandwidth. We then evaluate the accuracy of inline tightening, examining how to pick the best α for a variety of system sizes, probing errors, and bandwidth distributions. Finally, we deploy ThunderDome on live Planetlab hosts to test its performance in realistic wide-area settings.

6.1 Methodology

Simulation Methodology: Our simulator assumes a hub-and-spoke network model, assigning each host an upload and download bandwidth using one of the distributions described in the following paragraph. Each simulated time step corresponds to a tournament round. During each round, nodes perform pairwise bandwidth exchanges. The simulator distorts the results of these exchanges according to a probing error function. Ideally, this function would be guided by an empirical distribution of probing errors. However, to the best of our knowledge, none of the prior work on bandwidth estimation [10, 11, 13, 19, 21, 20, 25, 17] has provided a rigorous statistical characterization of the probing error. Generating such a characterization is beyond the scope of this paper. Thus, in our simulations, we assume that the fractional probing error is Gaussian with a mean of zero; we vary the standard deviation of the probing error to explore the impact of inaccurate probes upon our final upload estimates. All data points in the simulation graphs represent the outcome of 5000 trials.

Bandwidth Distributions: To evaluate our system, we employ two joint distributions of host bandwidths. Our *empirical distribution* attempts to capture the real world as best we can. In this distribution, 30% of peers are dial-up hosts and 70% are broadband clients [6]. For dial-up hosts, we assume V.92 transfer speeds of 53.3 kbps downstream and 48 kbps upstream [12]. For broadband hosts, we use a data set provided by DSL Reports [4], a web site that allows American broadband users to measure their upload and download speeds. The marginal empirical distributions for these speeds are shown in Figure 7(a).

Algorithm 8 Distr. Tighten ThunderDome—Initialization

Input: Set of n hosts $\{h_i | i \in [1..n]\}$
  Count k of rounds to run
  Set of $n/2 \cdot k$ coordinators $\{c_i^r | r \in [1..k], i \in [1..n/2]\}$
Output: Assignment of c_{init} for each host
  Assignment of c_{win} and c_{lose} for each coordinator

for all $i \in [1..n/2]$ **do**
  Set $j := i + n/2$
  Set $h_i.c_{init} := c_i^1$
  Set $h_j.c_{init} := c_i^1$
end for
for all $r \in [2..k]$ **do**
  for all $x \in [0..2^{r-2}]$ **do**
    for all $y \in [0..n/2^r]$ **do**
      for all $z \in [0..1]$ **do**
        Set $i := n/2^{r-1}x + y + 1$
        Set $s := n/2^r$
        Set $c_{i+sz}^{r-1}.c_{win} := c_i^r$
        Set $c_{i+sz}^{r-1}.c_{lose} := c_{i+s}^r$
      end for
    end for
  end for
end for
Set $c_i^k.c_{next} := \perp$

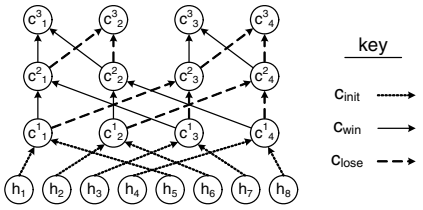


Figure 6: Topology of distributed algorithm with inline tightening for $n = 8$. Note the visual isomorphism to Figure 3.

Our *analytical distribution* captures the salient features of our empirical distribution, but it allows us to vary key parameters. The analytic distribution consists of a δ-fraction of dial-up hosts and a $(1 - \delta)$-fraction of broadband hosts. Although it would be straightforward to vary the speeds of dial-up hosts, such variation does not significantly change our results. Thus, we preserve the dial-up speed values from the empirical distribution. For broadband hosts, we generate two independent random variables. U represents the distribution of each host's upload bandwidth, and R represents the distribution of the ratio of each host's download bandwidth to its upload bandwidth. U and R follow lognormal distributions with parameters μ_U, σ_U, μ_R, and σ_R. To preserve directional asymmetry, R is lower-bounded[4] to unity. Random variable D, which governs each host's download bandwidth, is then computed as the product of random variables U and R, i.e., $D = U \cdot R$. Because the product of lognormals is lognormal, the marginal distribution of D is a lognormal distribution with parameters $\mu_D = \mu_U + \mu_R$ and $\sigma_D = \sqrt{\sigma_U^2 + \sigma_R^2}$. When fitted to our empirical distribution ($\delta = 0.3$, $\mu_U = 6.6$, $\sigma_U = 0.68$, $\mu_R = 1.4$, $\sigma_R = 0.49$), the marginal analytical distributions are shown in Figure 7(a).

[4]We use the rejection method [14] to establish this lower bound. For the values of μ_R and σ_R that we consider, less than 0.2% of random samples are affected.

6.2 Validating the Network Model

ThunderDome assumes that the network core has infinite available bandwidth, and that congestion occurs at the edge links connecting stub networks to the core. If this assumption is true, then given two peers separated by the wide area, the maximum transfer speed should be the minimum available bandwidth along the edge links connecting the two peers. A slower transmission rate should induce few packet losses at the edge or inside the core.

To test this hypothesis, we examined transfers speeds on the Planetlab testbed [24]. To provide ground truth about edge bottleneck speeds, we enforced bandwidth limits using a simple UDP rate limiter. Like the Trickle bandwidth shaper for TCP [8], our limiter intercepted calls to `send()` and `recv()` using link-time library interpositioning. The limiter wrapped these calls inside code which enforced traffic caps. Caps were defined with respect to a sliding bandwidth window of 500 milliseconds, i.e., nodes were limited to sending a maximum of b bits over any 500 millisecond period. We measured transfer speeds using bulk UDP transfers.

Figures 7 (b,c) shows the transfer speeds measured by receivers as a function of the upload caps on the senders (the receivers did not have download caps). The LAN results came from transfers between hosts in the same domain. The WAN numbers represent inter-domain transfers amongst the same host population. Host pairs were drawn from 32 different domains, and each data point represents 50 trials.

Figure 7(b) shows that mean upload estimates in the WAN were close to those in the LAN for target caps of up to 768 kbps. In both transfer scenarios, these estimates were biased upwards. However, this was partially an artifact of our traffic shaper [5]. In both the LAN and the WAN, estimation accuracy declined for upload caps above 768 kbps. For these larger caps, hosts began to hit the real bandwidth limitations imposed by the PlanetLab system. Each experiment on a PlanetLab host receives a fraction of the host's raw bandwidth, and (during the time of our experiment) senders could not reliably secure more than 768 kbps of upload bandwidth. This is indicated by both the falling mean estimate of upload speed, and also the increasing standard deviation of these estimates.

Across all upload caps, the WAN estimates had a larger standard deviation than the LAN estimates. This shows that our network model is not completely accurate. Even if the core has extremely high bandwidth, it still provides additional opportunities for packet reordering and delay. Regardless, these experiments show that the routing core can transmit at least 768 Kbps of PlanetLab traffic without generating congestion-induced drops. This demonstration is sufficient to validate the experimental results in Section 6.6. However, additional research is needed to determine whether middle mile congestion [18] is becoming more prevalent. Widespread middle mile congestion would break ThunderDome's pairwise exchange technique, which assumes that different exchanges involving the same host reveal bandwidth constraints bound to that host instead of to the unique paths connecting that host to its peers.

[5]Since it did not enforce burst limits at a granularity smaller than 500 milliseconds, the sender released clumps of packets at 500 millisecond intervals. From the receiver's perspective, the last packets in the last clump arrived earlier than they would have if the sender had a non-bursty transmission rate. These early arrivals nudge the receiver's bandwidth estimate upward.

6.3 The Threat of Underestimation Bias

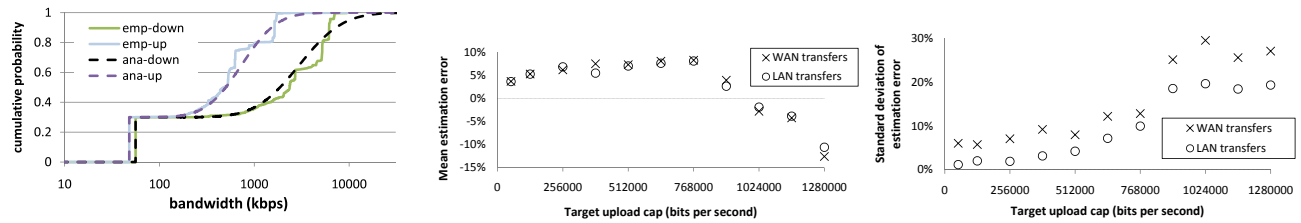
As shown in Section 4, measurement errors in bandwidth probes can wreck havoc on the resulting estimation accuracy if left uncorrected. Figure 8(a) depicts the simulated execution of our basic algorithm for a probing error $\sigma_{probing}$ of 20%. For each system size, the algorithm performed all $\log n$ rounds. The y-axis shows the fraction of hosts whose estimation error was greater than $2 * \sigma_{probing}$; for a given round, a host's upload estimate was either the losing bandwidth from a previous pairwise exchange, or the largest upload speed that it had observed up to that point. Looking at Figure 8(a), we see that for each value of n, estimation accuracy was essentially frozen after three rounds. Figure 8(b) explains why. In this graph, each curve represents the PDF for estimation error at the end of a particular round (the system size was held constant at 1024 hosts). Given that the probing error is Gaussian with a mean of zero, one might hope that the estimation error would follow a similar distribution. However, estimation error actually follows a bimodal distribution. The probability mass to the left represents the victims of underestimation bias, and the mass to the right represents hosts whose estimation error is governed by the (less pernicious) distribution of probing error.

The gray curve in Figure 8(b) depicts final estimation accuracy when we performed a single round of tightening as described in Section 4.2. Each node had two samples at best, so we did not employ α-tightening. Instead, we revised an estimate upward if the measurement from the tightening round was at least 1.5 times larger than the current estimate. Figure 8(b) shows that this scheme effectively eliminated underestimation bias, turning a bimodal distribution of estimation error into a normal one which approximated the distribution of probing error. Unfortunately, this tightening scheme required an additional round of bandwidth probes and a centralized controller to generate the schedule. In the rest of the evaluation section, we focus on the inline tightening procedure, which is much more decentralized.

6.4 Inline Tightening

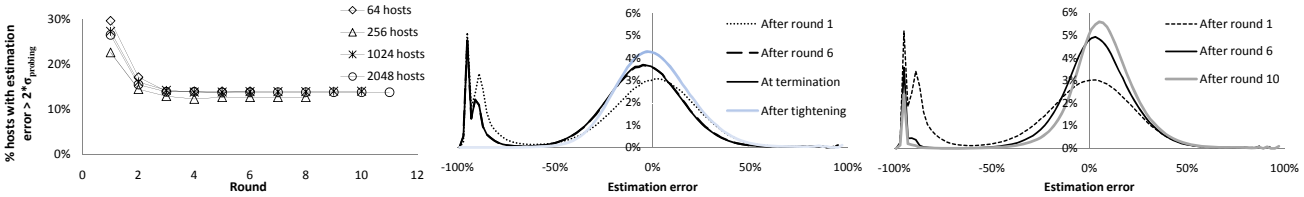
Inline tightening allows nodes to revise their upload estimates as the tournament unfolds. Figure 8(c) shows an example of this progressive refinement. As more rounds unfold, more victims of underestimation bias are identified, and their estimates are revised upward. Compared to the centralized tightening described in Section 4.2, inline tightening leaves more underestimated nodes. This is because the centralized algorithm exploits global knowledge of all upload estimates. By scheduling revision pairings according to estimation rank, the centralized algorithm increases the likelihood that an underestimated peer is discovered. The decentralized inline scheme lacks global knowledge and simply pairs winners with winners and losers with losers.

Despite the slight decrease in estimation accuracy, inline tightening remains attractive because of its speed. Compared to the centralized scheme, it removes an additional round of bandwidth exchanges. It also supports progressive revisioning, which is useful if the system lacks the time to run all $\log n$ rounds, e.g., because an online game wishes to place an upper bound on the wait time before the game begins. Given all of this, a natural question arises: which α provides the best estimation accuracy? Experiments show that the best α is a function of the number of rounds and the



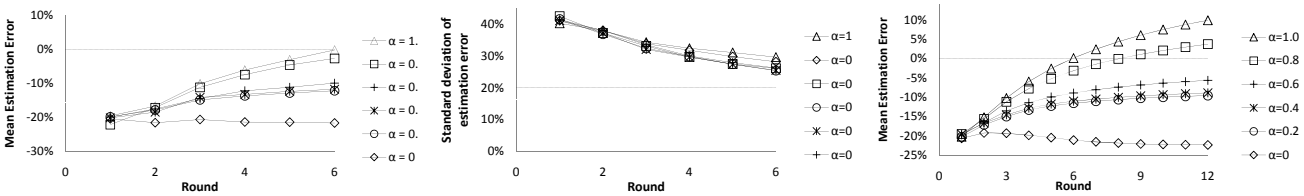
(a) Bandwidth distribution (b) Error mean (c) Error standard dev.

Figure 7: Network model validation: bandwidth (left); LAN transfers versus WAN transfers (middle, right).



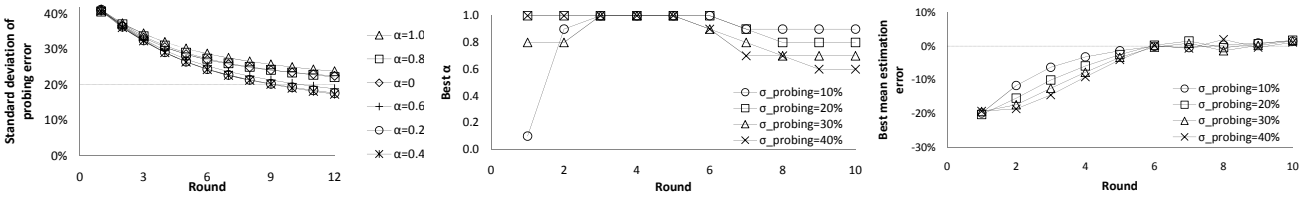
(a) Accuracy versus round number (b) PDF without inline tightening (c) PDF with inline tightening

Figure 8: Underestimation bias with and without inline tightening ($n = 1024$, $\sigma_{probing} = 20\%$, $\alpha = 0.8$).



(a) Error mean, 64 nodes (b) Error standard dev., 64 nodes (a) Error mean, 4096 nodes

Figure 9: Error estimation versus network size in a 64 node (left, middle) and a 4096 node system (right).



(a) Error standard dev., 4096 nodes (b) Selecting the optimal α (c) Error mean for ($\sigma_{probing} = 20\%$)

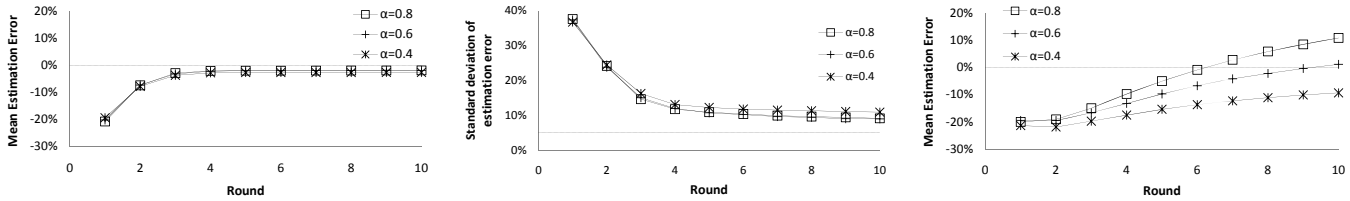
Figure 10: Error estimation versus network size in a 4096 node system (left). Selecting the best α as a function of rounds and $\sigma_{probing}$ (middle). For $\sigma_{probing} = 20\%$, 6 rounds is enough to push mean estimation error to 0 (right).

measurement error. Interestingly, it is *not* a function of the system size. For example, Figs. 9(a,b,c) and Figure 10(a) show estimation accuracy as a function of rounds for two systems, one with 64 nodes and another with 4096. $\sigma_{probing}$ was set to 20% in both systems. Comparing round 6 in Figure 9(a) to the equivalent round in Figure 9(c), we see that the performance of the various α's was virtually equivalent in both systems. This is because, after round k, two systems with different sizes will have performed the same number of pairwise exchanges, and thus produced the same number of samples over which to average.

Generally speaking, the fewer rounds that one is willing to expend, the higher α must be to push the mean estimation error to zero. However, depending on the probing error, there may be no α which can do this in a small number of rounds. For example, Figs. 9(a) and (c) show that in a network with $\sigma_{probing} = 20\%$, even the largest α cannot push the mean estimation error to zero in less than 6 rounds. Furthermore, Figs. 9(b) and 10(a) show that early termination leads to higher standard deviations for estimation error. Running for too few rounds prevents some of the dramatically underestimated hosts from pairing with high bandwidth peers and revising their estimates.

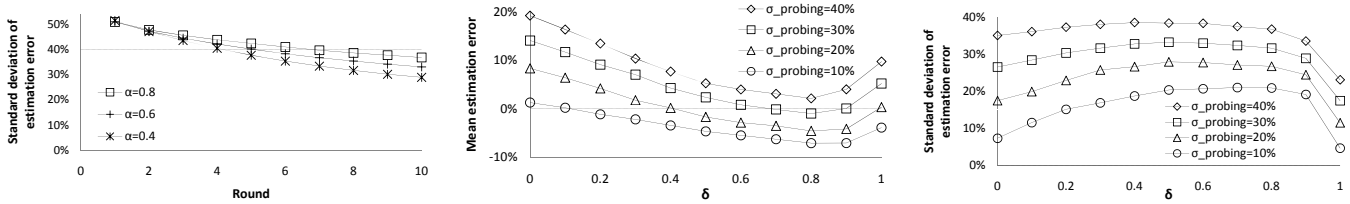
Figs. 11(a,b,c) and 12(a) hold the system size constant at 1024 hosts, varying $\sigma_{probing}$ from 5% to 40%. For low values of $\sigma_{probing}$, most α values provide equivalent performance. As $\sigma_{probing}$ increases, estimation accuracies diverge, making it crucial to pick a high α if fewer than $\log n$ rounds will be completed, but a smaller α otherwise.

Figs. 10(b,c) provides a concise summary of the results in this section. Figure 10(b) shows that high α are better for early termination, whereas moderate ones are better if most of the $\log n$ rounds will be performed. However, Figure 10(c) demonstrates that even an α of one cannot provide high estimation accuracy if too few rounds are completed.



(a) Error mean, $\sigma_{probing} = 5\%$ (b) Error standard dev., $\sigma_{probing} = 5\%$ (c) Error mean, $\sigma_{probing} = 40\%$

Figure 11: Estimation accuracy versus probing error $\sigma_{probing}$.



(a) Error standard dev., $\sigma_{probing} = 40\%$ (b) Error mean (c) Error standard dev.

Figure 12: Estimation accuracy versus probing error $\sigma_{probing}$ (left). Estimation sensitivity to fraction δ of dial-up hosts (middle, right).

6.5 Sensitivity Analysis

Up to this point, our simulations have been driven by an empirical bandwidth distribution containing 30% dial-up hosts and 70% hosts from the DSL Report data set. To evaluate inline tightening under different bandwidth regimes, we now turn to the synthetic bandwidth distribution described in Section 6.1.

Figs. 12(b,c) show ThunderDome's estimation accuracy as a function of δ, the fraction of dial-up hosts. For all data points, the system size was 1024 hosts, α was 0.8, and the lognormal parameters were set to values derived from the DSL Report data. Looking at Figure 12(b), we see that mean estimation error is very sensitive to fluctuations in δ. For extreme values of δ, the network becomes more homogenous, containing a minority or a majority of dial-up hosts. In either situation, the potential for widespread underestimation bias is low, since dial-up hosts are unlikely to be matched with high-bandwidth peers. For less extreme values of δ, such infelicitous pairings becomes more likely, and the mean estimation error is pulled downward. Looking at Figure 12(c), we witness a similar degradation in the standard deviation of estimation error.

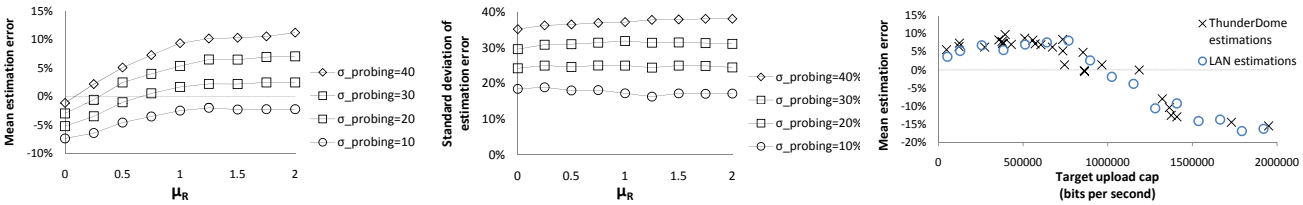
Figs. 13(a,b) show ThunderDome's sensitivity to μ_R, which controls the stretch between upload speeds and download speeds for non-dial-up hosts. In these experiments, we set δ to 0.3 and kept the other lognormal parameters set to the values described in Section 6.1. For high values of μ_R (i.e., high download-to-upload ratios), Figure 13(a) shows that underestimation bias rarely occurs within tournaments involving two lognormal hosts. As μ_R shrinks, lognormal hosts have increasingly smaller download-to-upload ratios, and underestimation bias emerges in tournaments between lognormal hosts. The worst-case underestimation error is still bound by the difference in upload speed between a dial-up host and a lognormal host. Thus, the standard deviation of the estimation error is relatively unaffected by μ_R.

The σ_R parameter controls the variation in download-to-upload ratios. Larger values of σ_R allow for a wider range of download-to-upload ratios, whereas smaller values imply more homogenous ratios. Due to space limitations, we do not show graphs for this sensitivity analysis. Instead, we note that for σ_R in the range 0.4 to 0.7, ThunderDome's estimation accuracy is insensitive to σ_R, with differences in estimation accuracy smaller than per-trial simulation variance. In this range, most download-to-upload ratios are sufficiently large that underestimation bias will not occur in tournaments involving two lognormal hosts. Thus, variations in σ_R have little impact on estimation accuracy.

6.6 ThunderDome over the Wide Area

For our final experiment, we deployed ThunderDome on the PlanetLab distributed testbed [24]. To provide ground-truth about available bandwidths, we capped transfer speeds using our custom rate limiter (§ 6.2). Caps were set using a bandwidth distribution of 30% dial-up hosts and 70% hosts from the DSL Report data. A special controller node acted as the scheduler, iteratively assigning hosts to tournaments and receiving the results of those tournaments. Nodes used inline tightening ($\alpha = 0.8$) to estimate their upload speeds, and the system ran for all $\log n$ rounds. Nodes resided in the same domains that were used for network validation in Section 6.2. In fact, each validation experiment was immediately followed by a ThunderDome tournament. PlanetLab hosts have extremely variable network and CPU loads, so running ThunderDome immediately after a validation test ensured that the validation results could identify faulty ThunderDome estimates. Note that the ThunderDome tests only involved WAN transfers, i.e., only one node from each domain participated in the tournaments.

Figure 13(c) shows a sample validation ThunderDome trial. The graph shows that ThunderDome's wide-area estimates have similar accuracy to local-area estimates. Essentially, ThunderDome on the wide-area did not add *new* estimation artifacts to those that already arose from our simple rate limiters and bandwidth estimators. In both the validation phase and the ThunderDome phase, bandwidth estimates diverged from target upload caps when the caps overshot the real limits imposed by PlanetLab's resource allocator. However, ThunderDome did not exacerbate this phenomenon, degrading in a similar fashion to local-area estimation.



(a) Error mean (b) Error standard dev. (c) Error mean

Figure 13: Estimation sensitivity to μ_R (left, middle). WAN ThunderDome accuracy ($n = 32$) in PlanetLab setting (right).

7. DISCUSSION

Given our results in Section 6, we believe that ThunderDome is an important practical step towards resolving the collaborative upload estimation problem. However, we realize that ThunderDome does not address potentially important aspects of the problem. For example, ThunderDome does not prevent peers from lying about their unidirectional bandwidths. Furthermore, although ThunderDome provides a fast measurement framework for estimating an unknown bandwidth distribution, it does not have a lightweight mechanism to determine the exact moment that non-stationary bandwidths change. Devising fast, efficient techniques to detect such phase transitions is an important area for future research. For example, one could imagine passively introspecting foreground traffic for changes in quality of service, or using truncated runs of the full unidirectional probes to identify potential network variance.

8. CONCLUSION

If a peer-to-peer system knows the upload bandwidth of each participant, it can reorganize its communication topology to improve scalability and performance [1, 2, 3, 9, 15]. Unfortunately, a straightforward application of current bandwidth estimators cannot reveal such upload constraints. ThunderDome is a new system for efficiently measuring these constraints in peer-to-peer systems. Using pairwise bandwidth exchanges as a fundamental measurement primitive, ThunderDome determines each host's upload bandwidth using time that is logarithmic in the total number of peers. ThunderDome uses statistical techniques to minimize the impact of probing errors. Simulations and a live PlanetLab deployment demonstrate that ThunderDome can produce accurate, fast estimates.

9. REFERENCES

[1] A. Bharambe, J. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In *Proceedings of SIGCOMM*, pages 389–400, August 2008.

[2] B. Biskupski, R. Cunningham, J. Dowling, and R. Meier. High-Bandwidth Mesh-based Overlay Multicast in Heterogeneous Environments. In *Proceedings of ACM AAA-IDEA*, October 2006.

[3] A. Bozdog, R. van Renesse, and D. Dumitriu. SelectCast—A Scalable and Self-Repairing Multicast Overlay Routing Facility. In *Proceedings of SSRS*, pages 33–42, October 2003.

[4] Broadband Reports. Broadband reports speed test statistics. http://www.dslreports.com/archive, October 29, 2008.

[5] R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, 27–28:297–318, October 1996.

[6] Communication Workers of America. Speed Matters: A Report on Internet Speeds in All 50 States. http://www.speedmatters.org/document-library/sourcematerials/sm_report.pdf,2007.

[7] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-Dispersion Techniques and a Capacity-Estimation Methodology. *IEEE/ACM Transactions on Networking*, 12(6):963–977, 2004.

[8] M. Eriksen. Trickle: A Userland Bandwidth Shaper for Unix-like Systems. In *Proceedings of USENIX Technical (FREENIX Track)*, pages 61–70, April 2005.

[9] J. Ghoshal, B. Ramamurthy, and L. Xu. Variable Neighbor Selection in Live Peer-to-Peer Multimedia Streaming Networks. Technical Report, University of Nebraska-Lincoln Department of Computer Science and Engineering, 2007.

[10] N. Hu and P. Steenkiste. Evaluations and Characterization of Available Bandwidth Probing Techniques. *IEEE Journal on Selected Areas in Communication*, 21(6):879–894, 2003.

[11] N. Hu and P. Steenkiste. Exploiting Internet Route Sharing for Large Scale Available Bandwidth Estimation. In *Proceedings of IMC*, pages 187–192, October 2005.

[12] International Telecommunication Union, Standardization Sector. ITU-T Recommendation V.92: Enhancements to Recommendation V.90, November 2000.

[13] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Proceedings of the Passive and Active Measurements Workshop*, pages 14–25, 2002.

[14] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[15] X. Jin, W. Yiu, S. Chan, and Y. Wang. On Maximizing Tree Bandwidth for Topology-Aware Peer-to-Peer Streaming. *IEEE Transactions on Multimedia*, 9(8):1580–1592, December 2007.

[16] K. Lai and M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proceedings of USITS*, pages 123–134, March 2001.

[17] K. Lakshminarayanan, V. Padmanabhan, and J. Padhye. Bandwidth Estimation in Broadband Access Networks. In *Proceedings of IMC*, pages 314–321, October 2004.

[18] T. Leighton. Improving Performance on the Internet. *ACM Queue*, 6(6):20–29, 2008.

[19] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proceedings of the GLOBECOM*, pages 415–420, November 2000.

[20] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, and R. Baraniuk. Multifractal cross traffic estimation. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement*, September 2000.

[21] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Proceedings of the Passive and Active Measurement Workshop*, March 2003.

[22] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, October 2008.

[23] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and K. Claffy. Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links. In *Proceedings of PAM*, March 2005.

[24] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using PlanetLab for network research: myths, realities, and best practices. *SIGOPS Operating Systems Review*, 40(1):17–24, 2006.

[25] J. Strauss, D. Katabi, and F. Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proceedings of IMC*, pages 39–44, October 2003.

[26] K. Vishwanath and A. Vahdat. Evaluating Distributed Systems: Does Background Traffic Matter? In *Proceedings of USENIX Technical*, pages 227–240, June 2008.

[27] WebSiteOptimization.com. *Average Web Page Size Triples Since 2003, http://www.websiteoptimization.com/speed/tweak/average-web-page/.*