Fast Scalable Robust Node Enumeration

Richard Black[1], Austin Donnelly[1], Alexandru Gavrilescu[2], and Dave Thaler[2]

[1] Micrsoft Research Ltd., 7 J.J. Thomson Avenue, Cambridge, CB3 0FB, UK
[2] Microsoft Corp., One Microsoft Way, Redmond, WA 98052, USA
`{rjblack, austind, alexang, dthaler}@microsoft.com`

Abstract. In a Local Area Network of computers, often a machine wants to learn of the existence of all the others satisfying some condition. Specifically, there are a number of existing discovery algorithms which permit an enumerator to reliably discover protocol participants, many of them idealised. This paper provides a new technique which controls the load placed on the network, minimises the time to completion, handles networks with significant loss, and scales over many orders of magnitude. Most significantly, the protocol also deals with the possibility of a malicious enumerator; an important contribution needed for current real-world networks. We also address the effects of several systems and engineering aspects, including scheduler jitter and clock quantisation.

1   Introduction

In a Local Area Network of computers, often a machine wants to learn of the existence of all the others satisfying some condition (such as being prepared to co-operate to perform a task). There are a number of existing discovery algorithms which permit an enumerator to reliably discover protocol participants. For example, various service discovery protocols (such as SSDP [3]) fall into this category, as do node discovery protocols (such as Browser [8]).

Some discovery protocols such as IGMP [2] do not wish to enumerate all the participants, but only wish to know whether there is at least one participant satisfying some property present. We do not consider this class of problem in this paper.

Closer to our problem is the case of link-local broadcast or multicast pings; they differ in that that they do not include reliability (though some of our techniques could perhaps be applied to that domain).

In this paper we consider the application in which enumeration of participants is initiated by a single *enumerator* broadcasting a Request message. In response to this, *responders* send a Response message, thus revealing their presence to the enumerator.

To avoid an implosion of traffic at the enumerator, a scalable enumeration mechanism requires a scheduling method whose purpose is to decide *when* the responder should send its Response. For example, the most common scalability method is a simple random delay within a fixed time interval.

We require a solution which also provides reliability (with high probability all nodes are enumerated), promptness (a small network takes a shorter time than a large network) and defence against a malicious enumerator. We say more about our requirements in

section 3.1, and comparison with previous work in section 2, but first we state more clearly what we mean by a malicious enumerator.

A malicious node on a local area network can engage in a number of extremely disruptive actions. It can consume all the bandwidth and prevent normal communication. It can also fake its source address in packets so as to reroute ("steal") traffic for another node. Given the possibility of such villainy it may seem pointless to discuss the operation of a protocol in the presence of malicious nodes. What we are attempting to defend against, however, is the incorrect apportionment of blame for bad behaviour on the network. Specifically, we wish to prevent a bad node from using a small amount of bandwidth to trigger good nodes to use a large amount of bandwidth. Our requirement is a protocol which cannot be tricked into causing the aggregate traffic load of good nodes to exceed the specified target rate.

In this paper we cover related work in section 2. In section 3 we introduce two load-adaptive algorithms which would be expected to have similar idealised behaviour: one which keeps a fixed probability of transmission and varies the periods over which it adapts to the load on the network, and one which operates on a fixed timescale but varies the probability. We evaluate these by analysis and simulation, showing that they behave differently in practice. Finally we recommend the preferred algorithm.

2   Related Work

There have been many examples of distributed load control for Media Access since randomised distributed media access was first implemented in 1970 in the Aloha system [1]. The most famous and commonly used is probably that in Ethernet, binary exponential back-off. Our work is dissimilar to controlling the load at the Media Access level; a MAC protocol is designed to arbitrate a single segment at full line rate, not a LAN comprised of many segments and where we want to consume only a small fraction of the available resource.

In work on scalable address allocation [5] and scalable reliable sessions [7], the problem under consideration was ensuring, regardless of whether there were a small or large number of potential responders, that a reasonable number (neither too large nor too small) responded to the request. Again our work is dissimilar in that the intention is that all responders should eventually respond, but at a rate which is below the rate directly achievable from the media in use.

Other work considers the problem of having agents in a network gathering and sharing information about each other in order to increase their knowledge of the network. Most packet routeing techniques work like this, either at a network layer directly, or at an overlay layer. There have been hundreds of publications in the field of *ad-hoc* sensor wireless networks addressing that problem which we do not attempt to list.

Our work is different in that all the agents are expected to be attached to substantially the same network, and only the enumerator is attempting to gather the information. In particular we address the case where one node's communication may interfere with another's and so the aggregate load must be controlled.

A specific example of more closely related work is resource discovery [6]. However, that work addresses the case where there is no centralised coordination point. In our

work we address the problem where there is a centralised coordination point, but the responders do not trust it to provide correct load control on the network.

Many other discovery systems (e.g., [6, 8]) assume that one node can be trusted to pass on the discovery of other nodes. Once again, our work is different in that the problem we address precludes this because (a) each node may have to include information with its response which makes responses too large to combine and (b) nodes may not trust each other.

Finally, Scalable Timers and RTCP [9, 4] have some similarities in that overall transmission of information is used to control each nodes transmission of information; however in those works each node sends many packets and the goal is a long term stability. In the enumeration problem each node is sending only once (unless retransmission is required) and is not attempting to find a long-term rate.

3   Enumeration Algorithms

In this section we give more detail on the requirements on our protocol. We then describe an idealised algorithm to use as a performance goal; it is impractical because it requires an oracle to be consulted to determine scaling information. Subsequently we describe our two practical algorithms which we will compare in section 4.

3.1   Requirements

To provide reliability, the enumerator must repeatedly retransmit its Requests. We do not wish responders to send a response to each request therefore the enumerator acknowledges Responses and responders only re-respond on receipt of a request without an acknowledgement. For efficiency, we assume that a large number of acknowledgements can be piggybacked with each Request by listing the responders' addresses in the enumerator's Request packet. Such a behaviour is illustrated in figure 1 where requests are being sent with a fixed interval T_E of 200 ms.

The corresponding behaviour of each Responder is illustrated in figure 2. The first request causes it to move to a pausing state; after some time it will send its Response and move to the sent state. In the sent state an acknowledgement causes it to transition to the done state, whereas a negative acknowledgement (a Request in which it is not acknowledged) causes it to return to the pausing state. A positive acknowledgement also causes a transition from the pausing state to the done state; this transition can occur when a response and a negative acknowledgement cross due to concurrency, followed by a positive acknowledgement.
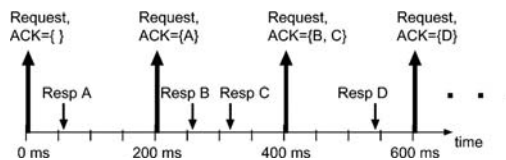

Fig. 1. Enumerator sends regular Requests, acknowledging Responses received in prior period

To provide timeliness, we desire that the enumeration complete as quickly as possible consistent with the desired network load (i.e., we do not want to statically limit the enumeration to proceeding at the slowest rate that might be needed for the largest possible network). Since the enumerator may be malicious we cannot (as many previous techniques do) trust it to control the rate at which responses are sent; hence one feature of the scheduling methods we present is that responders independently measure the load on the network due to the enumeration and use this to time their Response transmissions.

Specifically, we require that Responses are sent in such a way that they can be seen by all the Responders in the system and that by counting such Responses, Responders can execute a distributed load control function. In our descriptions below we describe the methods as using broadcast (or multicast) so that the responses can be counted, however variations are possible in which responders broadcast with some probability z and send directed (uni-cast) responses with probability $(1 - z)$. Provided z is not too small (there are still a statistically valid number of responses seen), it is still possible to estimate the overall number of responses by dividing the observed number by z. Such a change does not reduce the number of packets sent on the network, but it does reduce the number of broadcasts on the network which may be favourable. A potential disadvantage of such a change is that it implicitly assumes that the loss probability of broadcasts and unicasts is the same, which may not be true on wireless networks.

The enumerator stops transmitting Requests once there have been no Responses for long enough that there can be no more responders waiting to respond. Obviously any practical enumerator must defend against a responder which never ceases to transmit Responses, but that does not affect load control, and is not the aspect of malicious behaviour which we are considering in this paper.

Note that one cannot assume that retransmissions are initiated smoothly during operation; a malicious enumerator can arrange to withhold Request messages for a long time and then send a non-acknowledging Request which would cause many responders to be reactivated simultaneously.

Finally, there are several engineering requirements which are necessary for a practical implementation. First, the state retained by each responder must be strictly limited: it is impractical for them to keep a record of the address of the sender of every response they have seen, only the number of such responses seen. Second, we do not permit any significant computation other than increasing a counter when a response packet is received; computation is permitted only when a request packet is seen, or on a timer.
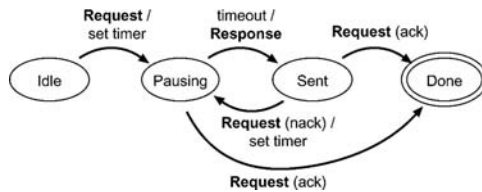

Fig. 2. A responder's state transitions

3.2   Notation

Assume M Response packets are each sent independently at a time chosen randomly and uniformly from an interval T, such that the average load per unit time over the whole interval is $\alpha = M/T$. Then the probability that the number of packets received in unit time, denoted X, has a value k can be approximated by the Poisson distribution:

$$P(X = k) = \frac{\alpha^k}{k!} \cdot e^{-\alpha}$$

For example, if $\alpha = 1$ the probability that a period of unit time has more than three packets is less than 2%, and the probability that it has more than four packets is about one third of one percent. This evaluation of α gives some degree of confidence that such random transmission rarely leads to bursts. We let $I = 1/\alpha$ be the average inter-Response time; hence $T = MI$.

Using q to represent the independent probability of loss on the network, the probability of a Response and a corresponding Acknowledgement both being received successfully is $(1 - q)^2$. The probability p of failure is thus $p = 1 - (1 - q)^2$.

Analytically, the number of Response / Acknowledgement exchanges required using a loss probability q (and combined Response / Acknowledgement loss p) until one of them is successful can be calculated. The expected number of exchanges for a single node is given by the mean of a geometric distribution with probability $(1-p)$ of success, which is $1/(1 - p) = 1/(1 - q)^2$.

We use N to represent the number of responders on the network; and $N_{\max}$ to represent the design maximum value for N. The expected number of exchanges for N nodes is thus $N/(1 - q)^2$ over a time $NI/(1 - q)^2$.

3.3   A Best-Case Bound Using Perfect Knowledge

We provide an approximation of a best-case bound on any practical algorithm by considering an impractical algorithm in which the enumerator consults an oracle to obtain the true number of responders on the network, and the responders trust that information.

The enumerator sends out the number of responders, N, in the initial Request message. Each responder schedules a Response at a random time distributed uniformly from the first round which is between 0 and $T_1 = NI$.

In the event of a negative acknowledgement (caused by loss) a responder does not retransmit until the next round (since other responders are still using the current round). Each Request message contains the number of responders still to be seen, hence a responder estimates the loss rate q on the network by comparing this with the initial value of N. As mention in section 3.2 above, if a similar loss rate applied to the Acknowledgements then the number of responders that will still be active can be estimated each round by multiplying N by $(1 - (1 - q)^2)$. This new value becomes the length of the next round in which to uniformly send a linear response.

This impractical algorithm minimises the amount of time needed to enumerate N nodes while maintaining the average load.

3.4   Successive Linear

This practical enumeration algorithm proceeds as a number of loosely-synchronised rounds. The ith round has a duration T_i, which is calculated from the result of the previous round. The first round has constant duration T_1, defined below. At the start of each round, every responder selects a random number: with probability ϕ it sends a Response (at a time chosen randomly, spread uniformly over the round's duration) during the round.

During a round, responders count Responses seen from other responders during the round; suppose the number seen by the end of the round is r_i. The expected value of r_i is given by

$$\mathbb{E}[r_i] = \phi \cdot R_i,$$

where R_i is the actual number of responders that were yet to send at the start of round i. Using r_i as the estimate $\mathbb{E}[r_i]$, yields an estimate for R_i. Of these R_i, approximately r_i have already transmitted, so the estimate of the number remaining at the start of the next round N_{i+1} is given by:

$$N_{i+1} = R_i - r_i \approx \frac{r_i}{\phi} - r_i$$

The responder needs to calculate T_{i+1}, the duration of the next round, so that it can either schedule its Response appropriately or wait the round out. We know that in this next round, on average ϕN_{i+1} responders will transmit a Response, therefore to space them out with average time interval I, we use a round duration of:

$$T_{i+1} = \phi N_{i+1} I = I\phi \cdot (\frac{r_i}{\phi} - r_i) = Ir_i(1 - \phi).$$

As each round progresses, r_i decreases and so too does T_i. When r_i gets below a threshold r_{min}, we decide there are sufficiently few responders on the network that further recursive subdivision is pointless, and that the remaining responders should send a Response at times uniformly distributed between now and $N_{i+1}I$.

The constant T_1 (time for the initial round) is calculated based on the worst case of $N_{\max}$ responders on the network. In the first round $\phi N_{\max}$ responders will respond, so to meet the desired minimum inter-Response time of I, we need to spread those responses over time $T_1 = \phi N_{\max}I$.

In the Sent state a responder continues to count Responses. When an unacknowledging Request message causes the responder to move back into the Paused state; its estimate of r_i is therefore the number of packets received since the start of the round in which it sent its response.

Suppose a malicious enumerator sends a negative acknowledgment after xI. For any nacked responder, the round in which it transmitted is ended either prematurely or extended. If it is extended then its r_i count will be larger than ϕR_i and so it will not overload the network. If it is ended prematurely then the x nodes will each have an r_i value of approximately x; thinking that there are $x(1/\phi - 1)$ nodes on the network they will each transmit with probability ϕ over their next interval giving rise to an additional relative load of $x\phi/(x(1/\phi - 1)) = \phi^2/(1 - \phi)$. This is not dependent on x, for small ϕ is small, and in practice the different periods caused by clock jitter make even this effect disappear. Successive Linear is therefore robust to a malicious enumerator.

3.5   Block Adjust

This practical enumeration algorithm is similar to Successive Linear except that instead of fixing ϕ and varying T_i we fix the time over which sampling is performed to a length T_b (the "block time"), and adjust the probability of transmission based on the number of responses seen in a previous round.

At the start of round i each responder has estimated that there are N_i responders left to transmit their responses. Every responder samples its random number generator and chooses a time uniformly distributed between zero and $N_i I$. If this is less than T_b then the responder sends its packet at the chosen time. If the time is greater than T_b then the responder does not send a packet in this round, but counts the number of responses seen during the interval T_b of the round, r_i and uses it to estimate N_{i+1} as follows.

If R_i is the true number at the start of the round then the expected value of r_i is:

$$\mathbb{E}[r_i] = \frac{T_b}{N_i I} \cdot R_i.$$

As before we can use r_i as the estimate $\mathbb{E}[r_i]$ and hence estimate R_i. Of these R_i, approximately r_i have already transmitted, so the estimate of the number N_{i+1} remaining at the start of the next round is given by:

$$N_{i+1} = \frac{r_i N_i I}{T_b} - r_i$$

Eventually some N_i will cause $N_i I \leq T_b$ at which point the responder will transmit in the current round and is finished. We also set N_1 to $N_{\max}$ to set the initial load.

Since a retransmission of a Response packet may be required, each responder continues network loading estimates until it has been acknowledged. In this way, the load due to retransmissions by other responders is continuously being monitored. Thus a value for N_i is always available, should a Negative Acknowledgement cause the Responder to return to the Pending state.

Recall that a malicious enumerator can negatively acknowledge a large number of responders simultaneously. This could cause a load of N/N_i in the extreme case so we deal with this by increasing the estimate of the number of responders by the worst case number which can have become unacknowledged by a Request. Specifically, every time a Request message arrives the number of Response messages received in total (since the Responder left the idle state) is sampled and stored; call this value N_{mb}. At the end of round i the current value of N_{mb} is compared against the value of N_{mb} at the end of round $i - 1$ (this value is stored in a further variable $_p N_{mb}$). If the value is greater ($N_{mb} > {_p N_{mb}}$) then the difference (equal to the worst case number of Responders that can have moved from Sent state to Pausing state due to Request messages in that round) is therefore added to N_i. Note that $_p N_{mb}$ is set on round boundaries and not when a Request message arrives. This adjustment value is an over-estimate for two reasons: first, the enumerator may have positively acknowledged some responders; second, some of the retransmissions from those responders may already have occurred in the previous round.

Several engineering changes are made to the Block Adjust method to take account of real life situations. First, to take account of the fact that jitter is not zero biased, the logic attempts to measure the actual duration T_a of the round (though we do not assume that

this has perfect accuracy). Secondly, even using T_a, hosts that are unlucky enough to get many successive large jitters may increasingly over-estimate the number of responders on the network. To stop any possibility of this getting out of hand the logic limits N_i to $100N_{\max}$. Thirdly, we are concerned about the effects of badly written device drivers which may cause packets to be dropped for a significant fraction of a block. Therefore we limit the reduction in the estimated number of hosts in any one round to a factor of three (approximately half an order of magnitude). The final estimator for N_{i+1} is:

$$N_{i+1} = \max\left(\frac{N_i}{3}, \min\left(100N_{\max}, \frac{r_i N_i I}{T_a} - r_i + (N_{mb} - {}_pN_{mb})\right)\right)$$

4   Results and Evaluation

We give an evaluation in this paper using a simulator written specifically for the purpose. This permits us to evaluate the technique under controlled loss, and jitter, and for networks much larger than can be constructed in the laboratory, in addition to comparing with the impractical technique.

The constants chosen for all the methods are designed to give an average load, α, of 1 packet per millisecond (thus $I = 1\text{ms}$). This makes them more easily comparable. We set T_E to be 200 ms, $N_{\max}$ to 10000, $r_{\min}$ to 4, T_b to 100 ms and ϕ to 0.1. We also assumed that time intervals can be measured accurately to a resolution of 20 ms. These are compatible with real world values.

It is unrealistic to expect perfect synchronisation, so for some experiments we introduce an extra delay to the time at which responders request to be woken, e.g., to transmit a Response or at the end of a round. This jitter parameter (which is applied randomly and uniformly) is intended to model sender OS-induced wakeup uncertainty together with variations in queueing delays from network and receiver OS; it is always additive (i.e., not zero-biased) because these effects can only ever delay an event, never result in it occurring early.

For loss we simulate receiver loss because that is pessimal for all the calculations (worse than realistic loss): a lost packet contributes to network load, but does not contribute to either (a) progress or (b) any load reducing control mechanism.

Several key aspects of the differences between the algorithms can be seen in figure 3 which shows the network load (averaged over 50 ms buckets and ten simulations) for the three algorithms with 3000 hosts, 100 ms of jitter, and 10% of loss.

- The Successive Linear algorithm uses well below the permitted load during the first 1000 ms due to its inability to adjust its behaviour before the end of its first period.
- The block adjust algorithm, during its first period T_b, transmits at the same rate (appropriate for a network of size $N_{\max}$) as Successive Linear, but rapidly achieves target load.
- Even the impractical algorithm takes a short time to achieve the target load because of the effect of jitter and loss on the network (e.g., because about 10% of the hosts do not see the initial request packet).
- The impractical algorithm has a dip in its usage of the network after 3000 ms; this is the effect of the jitter on its switch from the first phase (in which every node sends
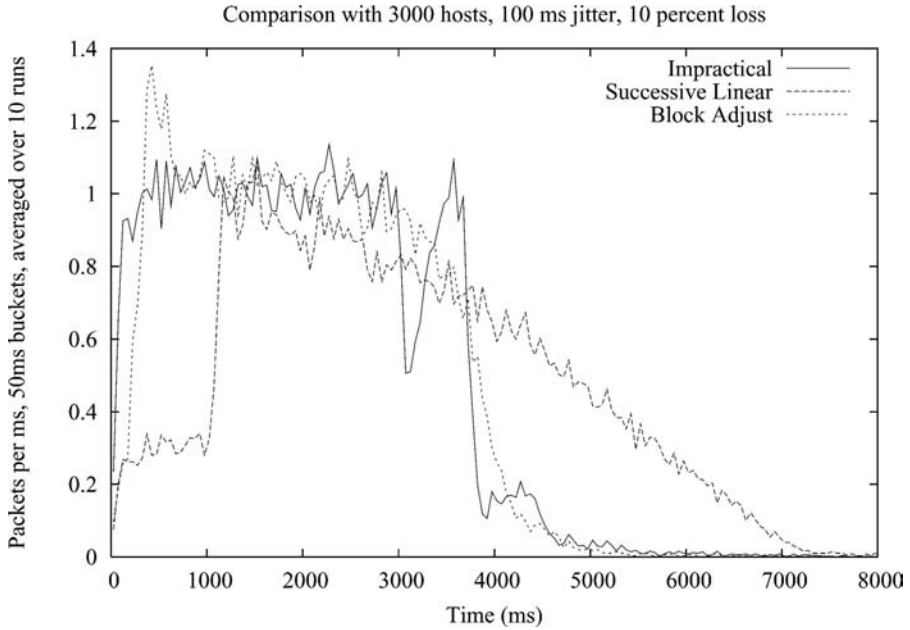
Comparison with 3000 hosts, 100 ms jitter, 10 percent loss

Fig. 3. Network load for 3000 hosts showing differences between algorithms

exactly once) to the second phase in which unacknowledged nodes retransmit after they believe the first phase to be over.

– Although Successive Linear briefly achieves the target load it has difficulty maintaining it and has a long tail. Since the jitter is not zero biased it effectively increases the sampling period; since the sampling period gets shorter in each round, the error increases in relative magnitude with the result that the method increasingly overestimates the load that is actually present.

– Using $N = 3000$, $q = 0.1$, and $I = 1$ ms in our earlier analytical result (without jitter), we get a completion time of 3703 ms, which is consistent with our simulated results here (which do include jitter).

– The block adjust algorithm has a small overshoot near to the start. This is an artefact caused by the large jitter. Since transmission times are delayed by up to 100 ms, the load during any particular period T_b is significantly affected by delayed transmissions from a previous period. This causes the under-damping; however as can be seen the method rapidly stabilises.

We also compared the methods at every half order of magnitude from 1 Responder to the design goal of $N_{\mathrm{max}} = 10000$ responders. To get an idea of how fragile the methods are we also tested them at half an order of magnitude more (30,000) responders; as can be expected the network load in the first period (T_1 for Successive Linear, and T_b for Block Adjust) was approximately three times the target, but subsequently the load returned to target.
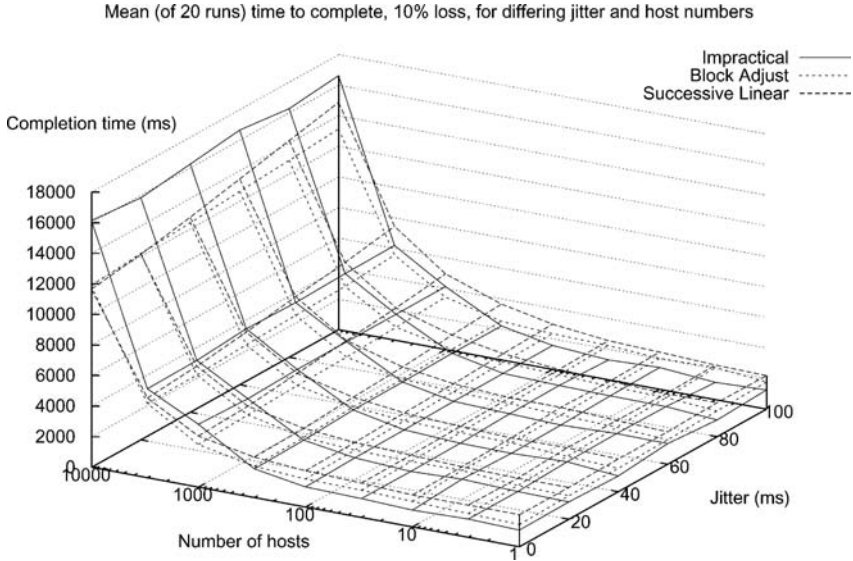
Mean (of 20 runs) time to complete, 10% loss, for differing jitter and host numbers

Fig. 4. Completion times

4.1   Completion Time

Figure 4 shows the finishing time for these methods (compared against the Impractical method) both for increasing numbers of hosts, and for increasing amounts of jitter.

It can be seen that with higher number of responders the finishing times become similar and dominated by NI. For fewer nodes, however, Block Adjust is superior because it can adapt more quickly; Successive Linear cannot begin to adjust until the end of the first time period at $\phi N_{\max} I$.

With higher jitter all methods take longer to complete because jitter is not zero biased. Nevertheless it can be seen that Block Adjust is less negatively impacted than Successive Linear.

The effect of loss on these methods can be seen by looking at the baseline data in figure 5. With higher levels of loss all methods take longer because retransmissions are required. Block Adjust retains its advantage over Successive Linear as loss increases.

4.2   Improvements

We originally described acknowledgements as being sent in the subsequent request packet. It is possible, however, to use any spare space within a Request packet to re-acknowledge previously acknowledged responders. The effects can be seen in table 1 by comparing the columns marked "repeat" with the original base-line (marked "none"). Since responders whose responses are received at the enumerator but where the (first) acknowledgement is lost now have additional chances of seeing an acknowledgement before retransmitting their response, the overall load on the network is reduced, leading to a quicker completion time.

Table 1. Time to completion showing several changes to the three algorithms

| % loss	Impractical				Successive Linear				Block Adjust			
	None	Repeat	Closer	Both	None	Repeat	Closer	Both	None	Repeat	Closer	Both
00	2228	2228	2166	2303	4746	4746	4701	4711	3054	3054	3020	2964
10	5334	4863	4775	4401	5880	5169	5500	4956	4269	3833	3773	3423
20	6573	5261	5738	4377	7952	5788	6849	5272	5596	4516	4740	3802
30	9002	6676	8026	5197	10336	6433	8365	5935	7719	5447	6050	4419
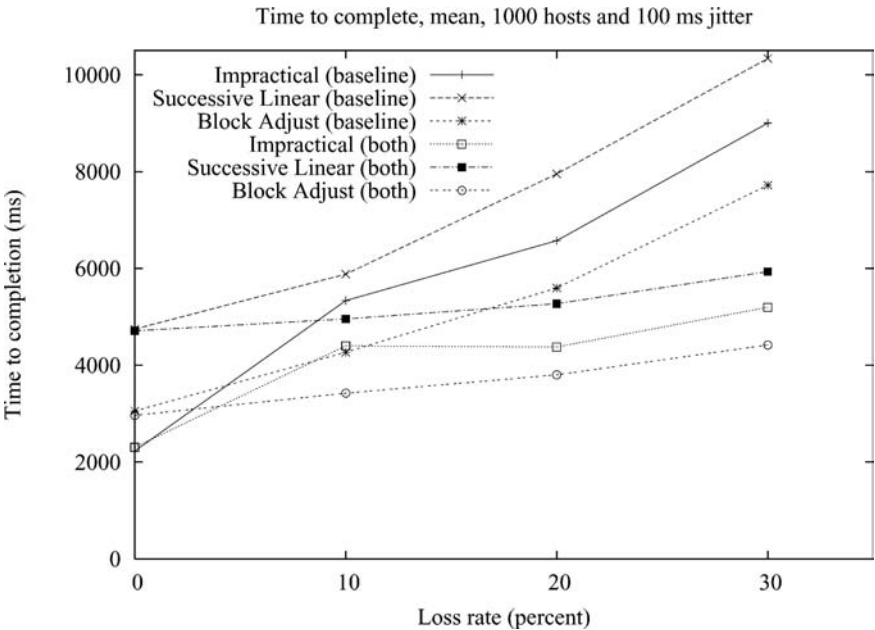
Fig. 5. Finish time showing both enhancements against the unenhanced case

The benefit of repeated acknowledgements is likely to be increased if a greater fraction of the request packet is carrying repeated acknowledgements (i.e., each response is acknowledged a greater number of times). This can be achieved by reducing T_E (since the number of new acknowledgements expected in each request is T_E/I). In fact, lowering T_E may also have a beneficial effect because even though more of the permitted overall load of the protocol on the network is consumed by the requests, the nodes which must retransmit responses discover this sooner; though they cannot do so until the load permits, the overall effect is that the tail is shorter, and completion is quicker. Merely reducing T_E from 200 ms to 100 ms so that requests are sent closer together in time has an effect shown in table 1 under the column "Closer".

Finally, the combined beneficial effect of both changes can be seen in the column header "Both". This is also shown graphically in figure 5.

5   Conclusion

We have introduced new techniques for discovery of nodes which are reliable, scalable, prompt, and robust to loss and jitter. Most importantly, our contribution is not susceptible to a malicious enumerator and cannot be provoked to overload a network. Of these the Block Adjust algorithm is found to be more responsive under realistic conditions.

We also show that using more bandwidth for repeated acknowledgements is overall beneficial to the protocol by reducing unnecessary retransmissions, even though it reduces the bandwidth available to Responses.

References

1. Norm Abramson. The Aloha System - Another Alternative for Computer Communications. In *Fall Joint Computer Conference, AFIPS Conference*, 1970.
2. B. Cain and S. Deering and I. Kouvelas and B. Fenner and A. Thyagarajan. *Internet Group Management Protocol, Version 3*. The Internet Society, October 2002. RFC 3376.
3. Yaron Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright. *Simple Service Discovery Protocol/1.0*. The UPnP Forum, http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt, version 1.0, draft 3 edition, October 1999.
4. H. Schulzrinne and S. Casner and R. Frederick and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. The Internet Society, January 1996. RFC 1889, see particularly section 6.2.
5. Mark Handley. Session Directories and Scalable Internet Multicast Address Allocation. In *SIGCOMM 1998*, volume 28(4) of *Computer Communications Review*, pages 105–116, October 1998. See especially section 3.1.
6. Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 229–237. ACM Press, 1999.
7. Roger Kermode and David Thaler. Support for Reliable Sessions with a Large Number of Members. In *Networked Group Communication*, November 1999. First International COST264 Workshop.
8. Microsoft Corporation. *Windows 2000 Server TCP/IP Core Networking Guide*, Appendix I - Windows 2000 Browser Service. Microsoft Press, May 2002. ISBN 0735617988, Also at http://www.microsoft.com/resources/documentation/windows/2000/server/reskit/en-us/tcpip/part4/tcpappi.mspx.
9. Puneet Sharma and Deborah Estrin and Sally Floyd and Van Jacobson. Scalable Timers for Soft State Protocols. In *Proceedings of the INFOCOM sixteenth annual joint conference of the IEEE Computer and Communications Societies*, page 222. IEEE Computer Society, 1997.