

# Scalable Local Area Service Discovery

Richard Black, Heimir Sverrisson and Laurent Massoulié  
Microsoft Research

**Abstract**—Existing methods for local area service discovery either don't scale or rely on a trustworthy directory server; in some environments these restrictions are unacceptable or impractical. This paper describes “Repeat-BAND” a generic method for service discovery that scales automatically without depending on any central component. The automatic scaling makes it fast on small networks and automatically load controlled on large networks, irrespective of the number of simultaneous discoveries taking place. It is generic in the sense that the automatic scaling technique can be applied to improve any particular service discovery system, and it is applicable across a large variety of types of network because we show how all the tuning parameters are derived. We present results showing controlled load discovery with scalability up to very large networks. We also show that the algorithms are simple and easy to implement; an important practical requirement since the method is used in Windows Vista and licensed by many hardware vendors. In addition, we consider the industrial requirement as to the certification of independent implementations, an aspect normally ignored in the academic literature.

**Index Terms**—service discovery, scalability

## I. INTRODUCTION

Service Discovery is an important part of a Local Area Network (LAN) such as Ethernet. On actively administered and provisioned networks large scale services such as Active Directory [4] provide this role. On networks without such infrastructure a distributed peer-to-peer approach is required which must be robust to the possibility of being misused to overload the network.

All the existing protocols have limitations (we describe several in section V); they operate on long time-scales (so are not responsive on small networks), or they do not scale to large networks, or they require a trustworthy directory server, or they implicitly rely on the Ethernet MAC's exponential backoff for load control (appropriate historically, but not for switched networks). The exception is the Block Adjust Node Discovery or BAND [2] method which is fast, scalable and robust but which supports only a single request on the network at a time.

In this paper we present a new technique for applying the load control mechanism of BAND to the general service discovery problem which may have many requests operating simultaneously over a network of up to 10,000 hosts.<sup>1</sup>

The main contribution of our system is that we regulate the load on the network; the completion time for a request is therefore dependent on the size of the network.

### A. The service discovery problem

A service discovery system is one in which an *enumerator* sends *requests* seeking to discovery particular services in the

network, and where *nodes* also known as *responders* or *hosts* send responses to the enumeration requests.

Existing and previous service discovery techniques have in the main focused on reducing the cost of the discovery on the *nodes* present in the network. For example, a typical technique has the nodes hash their property/name of interest and join a multicast group derived from the hash. An enumerator then sends a request to the group address and the responders send a response to its unicast address. The other nodes, not members of the group, are oblivious.

There are two fundamental problems with such an approach. First, there may be a very large number of nodes matching a particular enumeration causing an implosion of responses. Secondly, and much more insidious, it completely ignores the load on the *network* attributable to many such enumerations proceeding in parallel.

LANs have a bounded amount of bandwidth between nodes. Networks are comprised of fixed-sized switches which are interconnected with fixed-bandwidth links in the form of a spanning tree. Larger networks tend to be built out of larger switches, and smaller networks out of smaller switches, but large networks still tend to have a larger number of switches. In our experience a good rule of thumb is that a network with  $n$  nodes is comprised of  $\sqrt{n}$  switches each with  $\sqrt{n}$  nodes. The resulting spanning tree therefore has a depth of order  $\log n$ .

With this in mind we can consider the scalability of existing service discovery systems by considering the load on the links near the root of the tree. Any service discovery system in which responders send unicast responses to requests will cause the  $\sqrt{n}$  nodes below this link to respond to each of order  $n$  requests per unit time causing the load on the fixed sized link to grow at  $n\sqrt{n}$ . This clearly doesn't scale beyond small  $n$ .

In some systems responders multicast their response in such a way that a single packet is sufficient response for all currently active enumerations. Such systems cause the load on the fixed size link to grow at  $\sqrt{n}$ , a significantly better but still growing function.

In our design we started from the requirement that the load on every link in the network be fixed and not increase with the size of the network, giving scalability. Instead we permit enumerations to take a length of time which grows linearly with the size of the network.

We describe our design in section II, the results in section III and related work in section V.

## II. SCALABLE SERVICE DISCOVERY

Our “Repeat-BAND” technique has several interacting aspects which together provide efficiency, scalability and performance. First we explain the messages that are sent in

<sup>1</sup>Such large LANs are reported to exist at several locations.

our system and the state machines and information which is kept on each node. Then we explain the mechanism which responders use to determine when to send their responses. Finally we explain the additional behaviour of the enumerator.

### A. Messages and State Machine

As mentioned above, a scalable discovery system requires that response messages be sent as a multicast (or broadcast) so that although it travels across *every* link in the network, only *one* packet from the responder needs to travel across the link in order to satisfy the enumeration requests of *all* the current enumerators.

An enumerator may have to send multiple requests since packet loss might cause a single request to be missed at some points in the network. Responders therefore maintain a session table of known enumerators, and add an entry to the table, and initiate the sending of a response sequence when a new enumerator's request is observed. Repeat requests do not initiate another response sequence.

Session table entries are removed when the enumerator sends a completion indication, or after a suitable timeout. In addition, a sequence identifier is placed by the enumerator in its requests and stored by the responder in the session table so that if an enumerator aborts one enumeration session and starts another without a completion indication being seen at a responder then the responder can reactivate the state of the associated session.

To achieve resilience to packet loss it is necessary to retransmit response packets. Some systems transmit multiple times (e.g. SSDP), and some acknowledge packets and only retransmit when not acknowledged (e.g. BAND). We choose to do both, which increases both reliability and flexibility in an interesting way which we explain in section II-J.

When a responder creates a new session table entry it sets the retransmit count for this session (to e.g. 4). Once this number of responses have been sent the session is marked complete. The responses are sent spread out in a way determined by the load control mechanism and are not back to back (back to back packets do not have an independent probability of loss).

A responder also monitors requests from the enumerator to see if they contain piggybacked acknowledgements. If a responder is acknowledged by the enumerator then the responder can set the session directly to complete state. Transmission of responses ceases once all sessions are in complete state.

Using this scheme the usual behaviour is that the retransmission of the request from the enumerator, which also carries the acknowledgements for all the intervening responses, occurs before the load control mechanism permits retransmissions from the responders. Hence a single response is likely, however as loss rates increases the responders automatically retransmit more responses up to the specified limit.

### B. Adaptive Load Control

To achieve scalability we let each responder in the system regulate the load, rather than limit the rate at which requests may be sent. As in the BAND [2] method for node discovery,

the system operates with a target packet inter-arrival time  $I$ , a system constant derived from the expected packet size and the assumed smallest bandwidth link available on any network where the system may be used.<sup>2</sup>

If there are  $R$  responders (hosts) in the system the desire is to discover them all in a time equal to  $R \cdot I$ . Because  $R$  is unknown a control loop estimates its value by observing the behaviour of the network; all requests and responses are multicast by the system, so the arrivals at each responder can be used as input to the control loop.

Given an estimate  $N$  for the true number of hosts  $R$ , a responder calculates an evenly distributed random point in time in the range  $[0, N \cdot I]$  at which to transmit its response. As in BAND, the control loop divides time into blocks; at the end of each it evaluates the load on the network over the block and updates the estimate  $N$ .

If the length of the block over which packet arrivals are counted is  $\alpha$  times the packet inter-arrival time  $I$ , then the number of responses received  $r$  (in the lossless case) will follow the binomial distribution.

$$r = \text{Binomial}(R, \frac{\alpha}{N}) \quad (1)$$

Suppose the number seen in block  $i$  is  $r_i$ . The expected value of  $r_i$  is given by:

$$\mathbb{E}[r_i] = \frac{\alpha}{N_i} \cdot R_i. \quad (2)$$

where  $R_i$  is the number of active responders at the start of round  $i$  and  $N_i$  is the estimate at the start of round  $i$ . Using  $r_i$  as the estimate  $\mathbb{E}[r_i]$ , yields a new estimate  $N_{i+1}$  for the value of interest  $R_i$ .

$$N_{i+1} = \frac{r_i N_i}{\alpha} \quad (3)$$

We discuss the selection of  $\alpha$  below. Initially  $N_0$  is set to  $N_{\max}$ , the maximum number of hosts on a local area network.

It is straightforward to see that the recurrence estimate is self-stabilising. Suppose that initially  $N_0 = R$ . Let  $r_0$  be the random variable in the first block. The new estimate  $N_1$  will be  $r_0 R / \alpha$  and so the random variable in the second interval will be chosen from  $\text{Binomial}(R, \alpha / N_1)$ , the mean of which is  $\alpha^2 / r_0$ . This will lead to a new expected  $N_2$  of  $\alpha^2 / r_0 \cdot r_0 R / \alpha \cdot 1 / \alpha = R$ . Hence any excursion from the mean is automatically corrected in the subsequent block.

### C. Overload

The above binomial distribution could lead to periods where the random variable is larger than the mean. For a binomial distribution on  $n$  and  $p$ , the probability that the random variable  $X$  exceeds some constant  $c$  is given by the Chernoff bound [1]:

$$\mathbb{P}(X \geq c) \leq e^{-(c \log c / p + (n-c) \log \frac{n-c}{1-p} - n \log n)} \quad (4)$$

<sup>2</sup>It is straightforward to convert to a system where packets are not of fixed size, but we elide the details here to focus on the important aspects of our system.

In our system  $p = \alpha/n$ , and an interesting question is with what probability the load on the network will be more than double the target for the interval  $\alpha I$  (i.e.,  $c = 2\alpha$ ). In this case the probability is below one in a million for any  $\alpha \geq 36$  for  $N$  up as large as one million hosts.

There is a much more likely practical cause of overload. At any time a network with a discovery active could be joined to another one similarly active, leading to double the intended load. Such an overload should be detectable and the system should stabilise rapidly.

To deal with these two possible causes of overload we deliberately run the control loop to attempt to utilise half of the bandwidth allocated to the protocol.

A little additional care is needed in the choice of  $\alpha$ , since the Chernoff bound above does not consider the dynamic behaviour over multiple blocks. The fact that a low random sample leads to a correspondingly higher mean in the subsequent block causes larger tails of the distribution. We extensively simulated this for different values of  $\alpha$ , some of which are shown in Figure 1. The probability that the load exceeds 2 (our safety margin) is less than 0.001 for  $\alpha \geq 40$ .

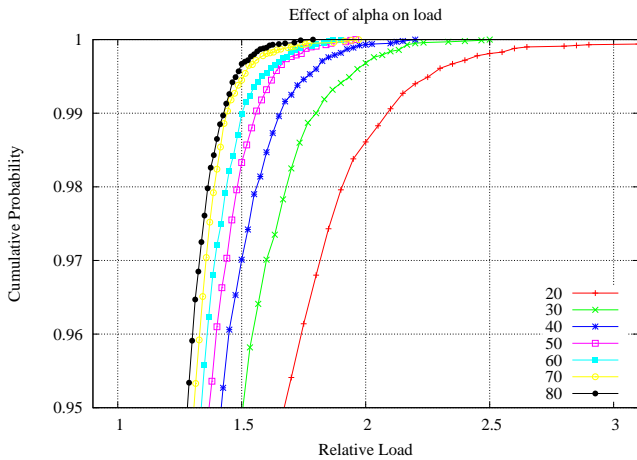


Fig. 1. Effect of  $\alpha$  on load.

#### D. Damping factors

There are several practical issues associated with equation (3) which must be addressed.

There is a non-zero probability that even in a large network a particular block may contain no transmissions; it is clearly inappropriate to set  $N_{i+1}$  to zero simply because  $r_i$  was zero; some damping is required.

There are also some real life problems. First, there may be clock jitter on the nodes and other scheduling delays which may lead to a reduced value of  $r_i$ , not because the network is small but because the nodes intending to transmit in the block were delayed in doing so. Second, device driver bugs and other serious networking errors can cause periods in which no packets are received, seriously skewing the estimator.

These two issues are quantified by an absolute time which is dependent on hardware and systems issues rather than the protocol. We model both the device driver bugs and the clock

latency by what we call *deaf time* (in practice, a value in the range 50-100ms is appropriate). This is time during which responses, though expected, are not in fact received. The load control loop therefore limits the possible reduction in the estimate  $N_i$  to that based on a deaf time's worth of responses in the block. Note that obviously a block time should be larger than the deaf time. We choose a parameter,  $\gamma$ , to represent the deaf time divided by the packet interval time (to make unitless).

As it is extremely unlikely that this limit will lead to an underestimate of  $N$  and (since this bound dominates the time to completion on small networks) it is possible to permit a temporary overload of the network by a factor  $\beta$  in such cases. Hence we require the following inequality:

$$N_{i+1} > \frac{N_i \gamma}{\beta \alpha} \quad (5)$$

This conforms to intuition: if the possible deaf time is small compared to the block time then it has minimal effect ( $\gamma$  is small compared to  $\alpha$ ), but as the possible deaf time becomes a more significant fraction of the block time ( $\gamma$  is not small compared to  $\alpha$ ) then we have less confidence in making large reductions in the estimator of the number of active hosts. Since the design handles a temporary overload of 2 due to networks being joined (see section II-C above), it is convenient to also set  $\beta = 2$ .

For an upper bound on  $N_i$ , we apply the same  $100N_{\max}$  bound of BAND.

The load control equation is therefore:

$$N_{i+1} = \max \left( \left\lceil \frac{N_i \gamma}{\alpha \beta} \right\rceil, \min \left( 100N_{\max}, \frac{r_i N_i}{\alpha} \right) \right) \quad (6)$$

#### E. Additional enumerators

A fresh request from a new enumerator could enter the network at any time. The load control mechanism must react to this, since a large number of responders will create new sessions and may initiate the sending of responses.

The naive response would be for all responders to reset their estimator to  $N_{\max}$ , however that would introduce excessive latency on a small network if several such new enumerations started in successive blocks: the estimates would not get small enough for a reasonable probability of responses being sent.

To deal with this we use the following approach. If a responder has seen at least one new enumeration request during a block time then it adjusts the  $N_{i+1}$  estimate at the end of the block after the standard load control calculation of (6). If  $N_{i+1}$  is less than half  $N_{\max}$  then it is doubled, otherwise if it is less than  $N_{\max}$  it is set to  $N_{\max}$ .

This solution has the advantage that the expected load in the next block will be at least 0.5 so progress is being made. In some circumstances it can lead to an expected load above target, but never beyond 1.5. It also has the advantage that responders still to respond to the earlier request have a smaller estimate and so are likely to respond sooner than the others. This helps earlier requests to complete earlier even when multiple enumerators are active on a network concurrently.

### F. Choice of parameters

The packet interval turns out to be the fundamental tuning parameter for deploying the algorithm in any setting. The packet interval comes from the expected packet size  $P$ , the capacity of the network  $C$  and the maximum fraction  $\lambda$  (between 0 and 1) of the network capacity to be used by the protocol. Then the packet interval  $I$  can be calculated as:

$$I = P/(\lambda \cdot C) \quad (7)$$

Due to the possibility of a partitioned network being joined in the middle of a discovery it is clear that  $\lambda$  should be less than 0.5.

The block length is  $\alpha$  times  $I$ , and the damping parameter  $\gamma$  is then deaf time divided by  $I$ . The value  $\beta$  can be chosen at will, subject to the constraint that  $\beta$  times  $\lambda$  must be less than one, in order to never overload the network:

$$\beta \cdot \lambda < 1 \quad (8)$$

Clearly the block length must be greater than the deaf time ( $\alpha > \gamma$ ). Additionally  $\alpha$  must be sufficiently large to gain a good statistical sample, however since the load control system can adapt only once per block it is advantageous for  $\alpha$  not to be too large.

We recommend  $40 \leq \alpha \leq 100$  subject to  $\alpha \geq 2\gamma$ . In practice on a network which might include 802.11,  $C$  is limited to 1 Mbps; in addition per-packet wireless overheads of approximately  $800 \mu\text{s}$  must be taken into consideration when choosing the packet size.

### G. Windows Vista LLTD

In the implementation of this work, which is found in Windows Vista, the choices for the various parameters were  $I = 6\frac{2}{3}$  ms, together with  $\alpha = 45, \beta = 2, \gamma = 10$ . This makes for a block time of 300ms. Four retransmissions are made. Since wireless networks are envisaged, these choices permit a packet size of about 366 Bytes. We will also use these values for presenting our results below.

### H. Stop criteria

In an un-scalable system, in which responses follow requests within a short fixed time period, it is obvious when the enumerator has seen all the responses. In our system the responses are spread over a time which is dependent on the unknown size of the network.

In the BAND system [2] there was a single enumerator so it could track the network load easily and use (5) to know when every possible responder must have responded. With multiple enumerators the network load may never decrease since new enumerators could regularly join the system.

We have several stop criteria. One comes from the simple requirement for each responder to reduce its estimate sufficiently to guarantee transmission. This can be calculated from the number of applications  $\psi$  of the recurrence (5) required to reduce the estimate  $N$  from its worst case value to below  $\alpha$ . This value can also be used as the elapsed time in which

if no new hosts are seen at the enumerator then it can stop. An additional optimistic stop criterion we use is to stop if the network load is low and no new hosts are seen over a enumerator sampling interval (defined below).

Our simulations show that these criteria do not miss responders and if new requestors are not entering the system the optimistic criterion is faster than the timeout criterion.

### I. Enumerator sampling interval

Enumerators have several objectives to achieve. They wish to send request packets so that all the responders on the network see them. They wish to send timely acknowledgements to reduce network load by reducing retransmissions from the responders, they must decide when the enumeration is complete (i.e. that they should stop listening for new responses), and they should avoid using up a significant fraction of the network capacity with their own packets which would reduce the fraction being used for carrying responses.

In our system we operate an estimation interval in the enumerator slightly differently that in the responder. The interval in the enumerator is scaled by the number of active enumerators on the network. Request and acknowledgment packets are sent once per interval which keeps the expected number of request packets per block time equal to one (we assume that acknowledgments are sufficiently small that a large number can be piggy-backed in a request; we return to this in section II-J below).

The minimum time that an enumeration should take even if no packets are seen is easily calculated from the damping behaviour of equation (5), since it controls the worst case time before a responder might send its response.

### J. Stealth

In our ‘‘Repeat-BAND’’ system, acknowledgements from the enumerators suppress repeated transmissions from the responders (provided all sessions are acknowledged); without acknowledgement the responders repeated transmission serve for reliability. A system using acknowledgements must have each enumerator acknowledge each responder. It is obvious that in a system with a very large number of enumerators compared to responders, that the bandwidth consumed by the acknowledgements could dominate.

Fortunately we can make a quantifiable choice as to whether enumerators should bother to acknowledge responders (as described above, many acknowledgements can be sent in a single packet). Suppose there are  $E$  enumerators / requestors, and  $R$  responders in the system. Also suppose that an acknowledgement is some small fraction  $f$  of the size of a response packet, and the count of retransmissions if a responder is unacknowledged is  $c$ . The transition point is therefore:

$$H + E \cdot H \cdot f = E \cdot c + H \cdot c \quad (9)$$

In our experience, an acknowledgement consumes a bandwidth equal to approximately one tenth the size of the response (e.g., the a minimum sized Ethernet frame and a MAC address, or a Web Services description and a XML encoded UUID), making  $f = 0.1$ . A system using repeats must use about  $c = 4$

repeats to have high confidence in seeing every responder. For large  $H$  it is thus better not to acknowledge when the number of enumerators  $E > 30$ .

Indeed, since each response satisfies every request, whether seen at the responder or not, it follows from the above discussion that it is possible for an enumerator to operate entirely in *Stealth* mode in which it sends neither requests nor acknowledgements, provided there are sufficiently many other active enumerations active on the network!

### K. Algorithms

Both algorithms begin by setting the various system constants that we have discussed.

Algorithm 1 describes the requestor functionality.

---

#### Algorithm 1: Requestor

---

```

1 Set  $I$  to target load;
2 Set  $T_E = \alpha * I$  ;
3 Set  $N_{max} = 10,000$  ;
4 Set  $N_{bound} = 100 * N_{max}$  ;
5 Set  $Requestors = 1$  ;
6 Set  $minLoad = 0.5 * \alpha$  ;
7 Set  $\psi = \log(N_{bound}/\alpha)/\log(\alpha * \beta/\gamma)$  ;
8 Set  $HostList$  as empty list;
9 Set  $minEnd = now + \lceil \psi \rceil * \alpha * I$  ;
10 repeat
11   Set counter  $r$  to zero ;
12   Set  $Start$  to now ;
13   Transmit a request acknowledging  $HostList$  ;
14   while  $now < Start + Requestors * T_E$  do
15     if see a response then
16       increment counter  $r$  ;
17       if new response then
18         add sender to  $HostList$ 
19       end
20     else if see a request then
21       increment counter  $r$  ;
22       if new requestor then
23         increment  $Requestors$ 
24       end
25     end
26   end
27   Set  $T_a$  to  $now - Start$  ;
28   Set  $load$  to  $r * Requestors * T_E / T_a$  ;
29   if ( $HostList$  has not grown and  $load < minLoad$  and  $now > minEnd$ ) then
30     Transmit a request acknowledging  $HostList$  ;
31     return  $HostList$ 
32   end
33 until forever ;

```

---

Line 7 calculates  $\psi$  as described in section II-H which is used for the timeouts.

The requestor starts by transmitting a request (line 13). This is safe because the load regulation is done by the responders.

When a new responder (host) is heard, its response is stored on the  $HostList$  (line 18). Duplicate answers need not be stored, and the size of  $HostList$  is monitored for the stop criterion in line 29.

Requests seen from other requestors are used in line 23 to update the number of requestors active in the system. All requests and responses heard during the wait interval are counted (in lines 16 and 21) and used to calculate the current load on the system (in line 28). The current load is compared to the  $minLoad$  and used as one stop criterion (in line 29).

In line 13 a request is transmitted. This means that a requestor will transmit once per sampling interval.

The requestor terminates in line 31 when the stop criterion becomes true.

Algorithm 2 describes the responder functionality.

---

#### Algorithm 2: Responder

---

```

1 Set  $I$  to target load;
2 Set  $T_b = \alpha * I$  ;
3 Set  $N_{max} = 10,000$ ;
4 Set  $N_{bound} = 100 * N_{max}$  ;
5 Set  $N = N_{max}$  ;
6 repeat
7   Clear newRequestor flag ;
8   Generate a random time  $T$  in  $[0 \dots N * I]$  ;
9   Set counter  $r$  to zero ;
10  while waiting for  $T_b$  to elapse do
11    if  $T < T_b$  then
12      at  $T$  during  $T_b$  begin
13        Send response ;
14        Decrement counter for each session ;
15        Set session complete if counter zero ;
16      end
17    end
18    if see a response then
19      increment counter  $r$ 
20    else if see a request then
21      increment counter  $r$  ;
22      if new requestor then
23        if acknowledges then
24          create complete session
25        else
26          set newRequestor flag ;
27          create new session with count 4 ;
28        end
29      else if acknowledges then
30        set session complete
31      end
32    end
33  end
34  Let  $T_a$  be the actual time waiting;
35  Set  $N$  to  $\max$  of  $N * r * I / T_a$  and  $N * \gamma / (\beta * \alpha)$  ;
36  if newRequestor flag is set then
37    if  $N < \frac{1}{2} N_{max}$  then Set  $N$  to  $2 * N$  ;
38    else if  $N < N_{max}$  then Set  $N$  to  $N_{max}$ 
39  end
40  Set  $N$  to  $\min$  of  $N$  and  $N_{bound}$  ;
41 until all sessions complete ;

```

---

As we described earlier the algorithm initialises the estimator to  $N_{max}$ , the pessimistic number of hosts in the system (line 3).

In line 8 we pick a random point in time,  $T$ , evenly distributed over the time we now estimate it will take for all hosts in the system to transmit with a spacing of  $I$ . If  $T$  lies within the  $T_b$  we will transmit (in line 13). Otherwise we just observe requests and responses in the system.

In line 35 we compute our estimate  $N$  based on the responses heard during the interval. This is the main load regulating part of our algorithm.

## III. RESULTS

We have done extensive simulations of the network load caused by our method. The simulator which we use is restricted to integer values for certain variables, therefore we use the following slightly different values from those described in

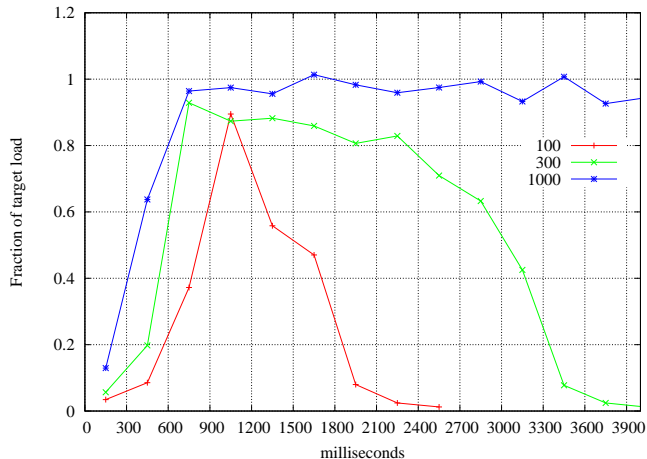


Fig. 2. Load for 100, 300 and 1000 hosts (truncated to 4 seconds).

section II-G above. We set  $I = 7, \alpha = 43$  making a block time 301 ms rather than 300 ms.

In figure 2 we show a basic enumeration for 100, 300 and 1000 hosts. The graph presents load data every 300 ms with an average of 50 simulations in order to smooth out the effects of the underlying random transmissions.<sup>3</sup>

The graphs show a characteristic start as the load control reduces its estimate from  $N_{\max}$  towards the true number of nodes on the network. This takes several block times, depending on the actual number of hosts. Finally the load has a characteristic tail as the estimator tends towards zero.

Note that during the “flat” portion of the graph, the load may not quite reach 1.0; this is because in these simulations the enumerator is not hostile and is correctly acknowledging the responses in each block. The number of active responders is therefore decreasing by approximately  $\alpha$  in each block in a way which cannot be safely assumed by the load control function, causing it to consistently slightly overestimate the number of active responders.

In Figure 3 we show a comparison of our Repeat-BAND method with the original BAND method on a network with high (20%) loss and 300 hosts. Note that the load shown on the vertical axis is the total number of packets *transmitted*, 20% of which are simulated lost randomly at each receiver, meaning that the control loop will be targetting a value which would appear graphically as  $1/0.8 = 1.25$ .

In this figure the benefit of the automatic retransmission in the Repeat-BAND case can be seen in the shorter and faster tail of the load. In the BAND original the responders do not retransmit until they see an implicit negative acknowledgement; if packet(s) from the enumerator are also dropped at the responder then it can be some time before the responders know to retransmit. By comparison in the Repeat-BAND system the responders will retransmit up to four times automatically.

In Figure 4 we show the effect of starting multiple enumer-

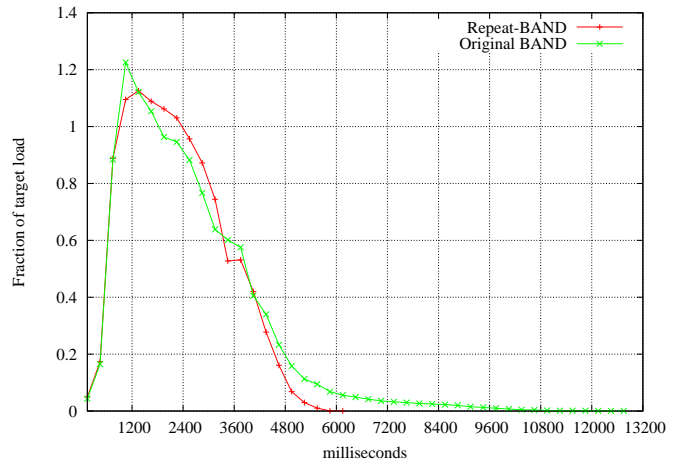


Fig. 3. Comparison with BAND at 20% loss, 300 hosts.

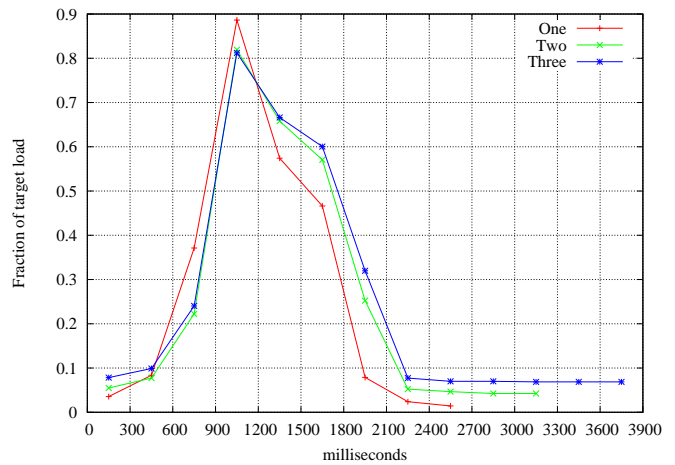


Fig. 4. Multiple enumerators starting together.

ators substantially together.<sup>4</sup> It can be seen that the network load is controlled and that the completion time is only slightly affected, due to the increased enumerator interval  $T_E$ .

The completion time of enumerators clearly depends on how many enumerators are active and when they enter the system compared to each other. As shown above, when enumerators start together there is very little effect. To illustrate the opposite effect, Figure 5 shows a network with 300 hosts (whose normal behaviour for a single enumerator was shown in Figure 2). In this case additional enumerators begin at multiples of 3000 ms (when the load caused by the previous enumerator is just beginning to tail off). The probability of each enumerator completing its enumeration is shown as a histogram overlaid on the load. Each new enumerator delays the previous enumerator by one enumeration span, but previous enumerators are mostly able to complete after this additional delay.

Our results clearly show that service discovery time with Repeat-BAND is approximately linear in the number of nodes

<sup>3</sup>The reader interested in confidence intervals or atypical behaviour is referred to the properties of the binomial distribution and the discussion in section II-C.

<sup>4</sup>They are slightly offset to avoid their correlated requests overlaying a small amplitude spiky pattern on the graph.

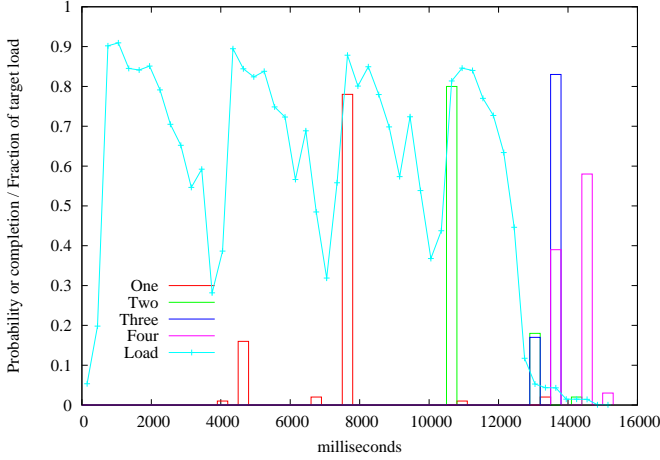


Fig. 5. Load and completion, four enumerators at 3 sec intervals.

in the network, by carefully controlling the load on the network throughout, and also highly efficient where multiple enumerators are present. Our completion times, especially in the case of packet loss, are superior to those of the previous best known BAND system.

#### IV. TESTING AN IMPLEMENTATION

When licensing technology to third parties to interact with a system such as this an interesting question arises which is usually ignored in publications. How does one automate the testing of a black-box implementation of such a system to see whether it sufficiently meets the licensing requirements. It is straightforward to test the state machine transitions, but testing the load control mechanism is more involved.

Let us imagine a controlled environment with an implementation and a test tool. The test tool can execute an instance of enumeration many times (including the generation of responses from other simulated responders) and record the times at which responses are observed. Response times are grouped into bins (of width  $\leq \alpha I$ ) and the probability of  $p_i$  exhibited by the system under test for bin  $i$  can be compared with the correct probability  $p_i^*$ . If the number of responses counted in bin  $i$  is  $b_i$  then this becomes a test that  $b_i \leq kq_i$  where  $k$  is the number of trials, and  $q_i$  is a threshold which can be derived from the probability of reaching false positive or false negative conclusions.

Considering a particular bin, we wish to check that  $p \leq Ap^*$  for some permissible deviation  $A$  which accounts for experimental error; and fail any implementation using  $p' \geq A'p^*$  for some suitable  $A'$ . We let  $\varepsilon$  be a bound on the probability of rejecting a good implementation (i.e. false negative); and  $\varepsilon'$  be a bound on the probability of accepting a bad implementation (i.e. false positive).

We can use the fact that the number of occurrences of an event with probability  $p$  over  $k$  trials is drawn from  $kp + \sqrt{kp(1-p)}\mathcal{N}(0, 1)$ .

The aim is to determine a minimum value  $k$  and threshold  $q \in [Ap^*, A'p^*]$ , so that:

$$\forall p \leq Ap^*, \quad \mathbb{P}\left(\mathcal{N}(0, 1) > \frac{k(q-p)}{\sqrt{kp(1-p)}}\right) \leq \varepsilon \quad (10)$$

$$\forall p' \geq A'p^*, \quad \mathbb{P}\left(\mathcal{N}(0, 1) < -\frac{k(p'-q)}{\sqrt{kp'(1-p')}}\right) \leq \varepsilon' \quad (11)$$

It is convenient to let  $\varepsilon = \varepsilon'$  and let  $t_\varepsilon$  denote the value of  $\mathcal{N}(0, 1)$  from tables. Hence:

$$\forall p \leq Ap^*, \quad \frac{k(q-p)}{\sqrt{kp(1-p)}} \geq t_\varepsilon$$

$$\forall p' \geq A'p^*, \quad \frac{k(p'-q)}{\sqrt{kp'(1-p')}} \geq t_\varepsilon$$

Equivalently,  $\forall p \leq Ap^*, \forall p' \geq A'p^*$ ,

$$p + \frac{t_\varepsilon}{\sqrt{k}}\sqrt{p(1-p)} \leq q \leq p' - \frac{t_\varepsilon}{\sqrt{k}}\sqrt{p'(1-p')} \quad (12)$$

$$p' - p - \frac{t_\varepsilon}{\sqrt{k}}\left(\sqrt{p(1-p)} + \sqrt{p'(1-p')}\right) \geq 0$$

A conservative test is:

$$p' - p - \frac{t_\varepsilon}{\sqrt{k}}(\sqrt{p} + \sqrt{p'}) \geq 0$$

This is minimised on  $p$  by setting  $p = Ap^*$ . Also we can look for  $\tilde{p}$  satisfying

$$\tilde{p} - p = \frac{t_\varepsilon}{\sqrt{k}}(\sqrt{p} + \sqrt{\tilde{p}})$$

Substitute  $x = \sqrt{\tilde{p}}$ ,  $\tau = t_\varepsilon/\sqrt{k}$ , and find the positive solution of the quadratic:

$$x^2 - \tau x - p - \tau\sqrt{p} = 0$$

$$x = \sqrt{p} + \tau$$

The condition of interest is determined at  $p = Ap^*$  by enforcing:

$$p' \geq \tilde{p}$$

Substituting  $\sqrt{\tilde{p}} = x = \sqrt{p} + \tau$  and rearranging for  $k$  gives:

$$\begin{aligned} p' &\geq (\sqrt{p} + \tau)^2 \\ \sqrt{p'} &\geq \sqrt{p} + t_\varepsilon/\sqrt{k} \\ \sqrt{p'} - \sqrt{p} &\geq t_\varepsilon/\sqrt{k} \\ k &\geq \left(\frac{t_\varepsilon}{\sqrt{p'} - \sqrt{p}}\right)^2 \end{aligned}$$

$$k \geq \left(\frac{t_\varepsilon}{\sqrt{A'} - \sqrt{A}}\right)^2 \frac{1}{p^*} \quad (13)$$

The smallest correct probability in our test, by giving the lower bound for  $p^*$ , together with our arbitrary slackness choices of  $A$  and  $A'$  can be used in (13) to directly give  $k$  the number of experiments required, and for each test a  $q$  can be chosen within the bounds provided by (12).

### A. Numerical example

Suppose we test an implementation using the parameters of Section II-G and that bins are the maximum width, the same as a block time ( $\alpha I = 300$  ms). The probability of a response in the first interval is  $p^* = \alpha/N_{\max} = 0.0045$ . It is convenient to choose  $\varepsilon = 10^{-4}$  in which case  $t_\varepsilon = 4$ . Allowing 20% experimental error but failing any responder with a probability more than double the target gives  $A = 1.2, A' = 2$ . Equation (13) tells us that  $k \approx 35000$  trials are required.

We can therefore use  $t_\varepsilon = 4, p^* = 45/10000, p = 1.2p^*, q = 2p^*, k = 35000$  in (12) giving:

$$243.8422 \leq q \leq 244.3272$$

Hence the decision threshold for this test is 244 events.

### V. RELATED WORK

Our Repeat-BAND method for local area service discovery adapts the distributed load control method set forth in the BAND system for node discovery [2]. Our main achievement is that we extend the system to support many simultaneous requests (many enumerators). We also extend the work to a much larger range of possible applications by examining the dynamic properties of the load control method and elucidating the fundamental tuning parameters.

Most traditional service discovery methods rely on a central element, i.e., a directory server, to distribute information. This is true of systems like JINI [14] and Salutation [9].

Work has been published on how the Ethernet could scale to a million nodes [11] where the authors publish measurements of ARP traffic in a network of 2500 hosts. They observe peak traffic more than ten times the average. They do not discuss regulating the transmission rate, as we are suggesting, instead they suggest a directory services solution.

The *Service Location Protocol* [6] works in two modes, either with or without a *Directory Agent* (DA). The DA is recommended for larger networks since load control is otherwise almost absent. In the system without a DA, a requestor uses multicast to ask for a service and responders pick a random time during the next three seconds to reply. This obviously does not scale to systems where there are thousands of replies to a request.

There are several methods that have been developed primarily for wireless low-power ad-hoc mobile networks e.g., Bluetooth [12], Konark [7] and DeapSpace [8]. These methods assume a small number of hosts in the system; hence they focus on minimising collisions and power consumption, rather than on controlling network load in a large system.

The IETF working group on *Zero Configuration Networking* [15] considered service discovery using multicast DNS SRV records [3]. The load control mechanism is rather simple: a requestor may retransmit a request after a second and subsequently doubles the retransmission interval. No load control is performed by the responders except for a small (500ms) random reply delay and use of *Known Answer Suppression* to stop a responder replying to the same request multiple times.

The UPnP Forum [13], a collection of several hundred companies, have a standardized service discovery protocol

called *Simple Service Discovery Protocol* (SSDP) [5] which uses HTTP over UDP multicast and has a combination of service announcements and discovery. SSDP has no load control and so does not scale beyond a small network.

An extensive literature survey of the current state of the art can be found in [10].

### VI. CONCLUSION

We have shown that our Repeat-BAND method for local area service discovery extends previous known work by permitting multiple enumerators to proceed efficiently in parallel. In addition, our method improves performance in the case of packet loss. By examining the fundamental interaction of the control parameters we have explained how the technique can be used generically on different network technologies and discovery problems. An additional contribution is an analysis which permits statistical black box testing of an implementation of our method.

### REFERENCES

- [1] Chernoff bound, June 2004. Available from: [http://en.wikipedia.org/wiki/Chernoff\\_bound](http://en.wikipedia.org/wiki/Chernoff_bound).
- [2] Richard Black, Austin Donnelly, Alexandru Gavrilescu, and David Thaler. Fast scalable robust node enumeration. In Raouf Boutaba, Kevin C. Almeroth, Ramón Puigjaner, Sherman X. Shen, and James P. Black, editors, *NETWORKING*, volume 3462 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2005.
- [3] Stuart Cheshire. Multicast dns, June, 2005. Available from: <http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt>.
- [4] Microsoft Corporation. Active directory, March 2003. Available from: <http://www.microsoft.com/windowsserver2003/technologies>.
- [5] 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\\_ssdv\\_v1\\_03.txt](http://www.upnp.org/download/draft_cai_ssdv_v1_03.txt), version 1.0, draft 3 edition, October 1999.
- [6] E. Guttman, C. Perkins, J. Veizades, and M. Day. Rfc-2608 service location protocol, version 2, June 1999. Available from: [citeseer.ist.psu.edu/wetherall98ants.html](http://citeseer.ist.psu.edu/wetherall98ants.html).
- [7] Sumi Helal, Nitin Desai, Varun Verma, and Choonhwa Lee. Konark – a service discovery and delivery protocol for ad-hoc networks. In *Third IEEE Conference on Wireless Communication Networks (WCNC)*, New Orleans, 2003.
- [8] Reto Hermann, Dirk Husemann, Michael Moser, Michael Nidd, Christian Rohner, and Andreas Schade. DeapSpace - transient ad hoc networking of pervasive devices. *Computer Networks*, 35(4):411–428, 2001.
- [9] The Salutation Consortium Inc. Salutation architecture specification, version 2.1, 1999. Available from: <http://www.salutation.org>.
- [10] Jay R. Moorman, John W. Lockwood, and Sung-Mo Kang. The state of service protocols. Available from: <http://citeseer.ist.psu.edu/moorman00state.html>.
- [11] Andy Myers, Eugene Ng, and Hui Zhang. Rethinking the service model: Scaling ethernet to a million nodes. In *ACM SIGCOMM HotNets*, 2004.
- [12] Bluetooth SIG. Bluetooth specification core version 1.2, November, 2003. Available from: [https://www.bluetooth.org/foundry/adopters/document/Bluetooth\\_Core\\_Specification\\_v1.2](https://www.bluetooth.org/foundry/adopters/document/Bluetooth_Core_Specification_v1.2).
- [13] The UPnP Forum, <http://www.upnp.org>. *The UPnP Forum*, 2005.
- [14] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999. Available from: <http://doi.acm.org/10.1145/306549.306582>.
- [15] A. Williams. Requirements for automatic configuration of ip hosts, September, 2002. Available from: <http://files.zeroconf.org/draft-ietf-zeroconf-reqts-12.txt>.