DIP: Distance Information Protocol for IDMaps

Yixin Jin Beichuan Zhang Vasileios Pappas Lixia Zhang
Computer Science Department
University of California, Los Angeles
{yjin, bzhang, vpappas, lixia}@cs.ucla.edu

Sugih Jamin
Department of EECS
University of Michigan
jamin@eecs.umich.edu

Abstract

The Internet Distance Map Service (IDMaps) [3] provides distance estimates between any pair of hosts connected to the Internet. The IDMaps system comprises two component types: Tracers that measure distances between IP address prefixes, and Servers that collect measurement results and answer distance queries. The Distance Information Protocol (DIP) is used for Tracers to report measured distance data to Servers. The dynamics on the Internet topology, the distributed nature of autonomous Tracers and Servers, and the vast size of the data set require that DIP provide highly adaptive and scalable data dissemination from Tracers to Servers. DIP is a soft-state announce/listen protocol and scales independently from the total amount of measurement data by all Tracers. DIP achieves its scalability through a combination of staged timers, positive feedback, and feedback suppression techniques, which enable DIP to disseminate only the most useful measurement data to Servers in a dynamic way. Simulations verified DIP's scalability and adaptability under various network conditions.

1. Introduction

The Internet Distance Map Service (IDMaps) [3] aims at providing distance[1] estimates between any two hosts on the Internet, so that applications can learn network distances quickly and efficiently without performing actual measurements themselves. Possible applications include locating the nearest mirror server, building efficient content distribution networks and P2P networks. As shown in Fig. 1, IDMaps has a two-layer structure. Tracers are light-weight measurement processes that are expected to be widely deployed on the Internet over time. Tracers measure network delay to different locations on the Internet and report the results to Servers. IDMaps Servers[2] collect measurement reports from all the Tracers and provide query/reply service to end-user applications by estimating network distances. Distance Information Protocol (DIP) is the component that glues these two layers together. It is used for reporting measurement results from Tracers to Servers and sending feedback messages from Servers to Tracers. The issues of how Tracers measure distances to various locations and how IDMaps Servers estimate the distance in answering applications' queries are addressed in [13] and [3], respectively; they are not part of DIP's functionality.

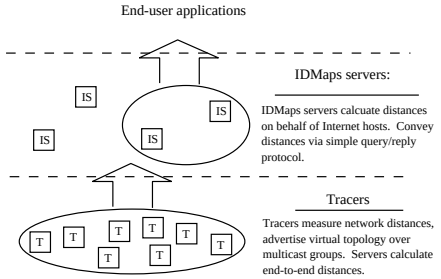


Figure 1. IDMaps Architecture

As a large-scale information dissemination protocol, DIP is designed as an announce/listen protocol. Each Tracer periodically advertises its measurement results to a multicast group comprising all Servers, while Servers just passively collect these reports. This soft-state approach gives the protocol adaptability and robustness needed in the dynamic operational environment, in which events such as network change, packet loss, node failure, adding or removing Tracers and Servers are not exceptional. Unfortunately, this simple "open-loop" announce/listen protocol does not work well in IDMaps system because our data size is huge. A soft-state protocol generally incurs extra transmission overhead due to periodic refreshment. It is not an issue when

[1] Throughout this paper, network distance refers to network latency, though IDMaps design can support other metrics such as bandwidth as well. See [3] for more discussion.

[2] In previous publications [3] [4] [6], we refer the same entity as "*Clients*" to emphasize that they consume information provided by Tracers. Throughout this paper, we will call them "*Servers*" to emphasize that they provide distance estimates to applications.

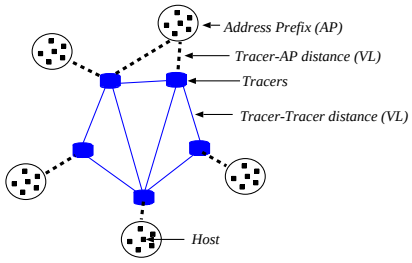


Figure 2. Distance Map

the data size is small, but will be prohibitively expensive when data size is as large as that in IDMaps system. Periodically sending all the data will either consume too much bandwidth or take too much time. Thus the challenge is to design DIP as a protocol scalable to data size and retain the advantages of soft-state approach as well.

Our main approach to scale DIP is information differentiation. Servers prioritize different data items based on their usefulness in distance estimation, then send the conclusion to Tracers as feedback. Based on the feedback information, Tracers are able to allocate more bandwidth to more useful data and less bandwidth to less useful data. When the most useful data is only a small portion of the entire data set as in IDMaps, the system can achieve great scalability and only lose little performance. To make this scheme work, we incorporate effective techniques to adjust sending rate and suppress duplicate feedbacks in our design. We believe that DIP design presents not only a scalable, robust and adaptive solution to IDMaps system, but also valuable insights on how to scale a soft-state protocol to large data size.

The rest of this paper is organized as follows. We begin with overview (Section 2) on IDMaps background and DIP's design rationales, then discuss protocol design in detail (Section 3), followed by performance evaluation (Section 4). We discuss related work (Section 5) and give a summary (Section 6) in the end.

2. Overview

2.1. Background of IDMaps

Since it is very difficult to keep measuring and reporting distances to every single host on the Internet, IDMaps groups nearby hosts into clusters represented by a common IP address prefix (AP). Hosts are grouped into APs according to network distances, as all hosts in one AP share relatively the same distance to Tracers. Thus Tracers can measure distance to just one host in an AP and use the result to represent the approximate distance to all the hosts in that AP. Though they take the same representation, IDMaps' AP is different from routing prefix since the latter is grouped according to routing policy. Hosts belonging to the same routing prefix may locate far away and have very different distances to Tracers.

A Tracer measures distances to other Tracers and APs, and reports the results to Servers. An IDMaps Server combines measurement results from all Tracers and build a *distance map* (Fig. 2) , which is a virtual network topology. A link in the distance map is called a *Virtual Link* (VL). When a client application (e.g., web browser) queries for distance between two hosts, the IDMaps Server will find out the corresponding APs and estimates the distance by calculating the length of the shortest path between these two APs in the distance map. Since the Internet topology changes from time to time, Tracers must perform measurements and send reports continuously. IDMaps' goal is to provide distance estimates with an update frequency on the order of days, or if necessary, hours. Such information only adjusts to "permanent" topology changes instead of transient network conditions. Instantaneous or near-instantaneous distance information is both impossible to be distributed globally and of diminishing importance to future applications [3].

2.2. Design Rationales of DIP

In IDMaps, there are a large number of both producers (Tracers) and consumers (Servers) of the raw distance information. Each Tracer produces a small portion of the distance map; Each server collects information from all Tracers and computes the entire distance map. The ideal mechanism for this many-to-many distribution is IP multicast because of its efficiency. IP multicast also provides an effective way to handle dynamic membership. At any moment, Tracers and Servers may join the system as deployment is in progress, or leave the system as old hardware/software is being phased out. Tracers and Servers are also distributed widely on the Internet under different administrative control. With IP Multicast, we can identify all the Servers by a multicast group G_S and all the Tracers by another multicast group G_t, eliminating the need for explicitly tracking every Tracer and Server. It allows each Tracer to transmit its small portion of the distance map without coordination with other Tracers. Tracers never need to know about the Servers in advance and vice versa. In this sense, IP multicast serves as a valuable discovery mechanism as well as a distribution mechanism. In the case that IP Multicast is not available, IDMaps can use end-host based multicast schemes (e.g., [16], [17], [12]) to obtain the same functionality.

There are two basic approaches to send raw distance information from Tracers to Servers: hard-state and soft-state. In the hard-state approach, Tracers report their distance information by reliable delivery. After the initial delivery, only significant updates will be reported. Therefore the transmission overhead is kept small. However, as in an

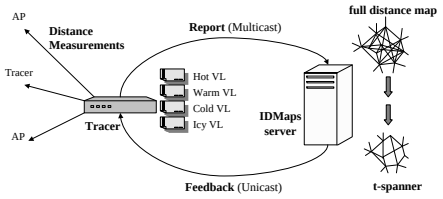


Figure 3. Overview of DIP operations

Internet-wide distributed system like IDMaps, no component is guaranteed to be error-free: packets may be dropped, links may go down, Tracers and Servers may crash etc. The hard-state approach has to prepare for all possible errors and handle them explicitly. Furthermore, reliable delivery is not always necessary, because when a new measurement result is available, whether or not the old result is delivered becomes irrelevant. We adopt the soft-state approach because it simplifies protocol operations and makes the protocol adaptive to unexpected errors or changes. In a soft-state protocol, Tracers periodically send distance information as refresh messages to Servers using best-effort transmission. Over time, a Server will receive all the VLs. Each VL has a time-to-live which is set to be several times longer than the transmission period. Servers simply delete any raw distances whose age exceeds the assigned time-to-live.

However, periodic refresh messages between Tracers and Servers introduce more transmission overhead, which is proportional to the data size. In the worst case, if each Tracer traces every other Tracer and all the APs, the data size is on the scale of $N * N + N * P$, where P is the number of APs and N is the number of Tracers. As shown in Section 3, our preliminary results suggest that $P \cong 300K$ and $N \cong 3K$, so the total data size can be hundreds million, or even thousands million in the worst case. Periodically sending and receiving all these information will consume too much bandwidth, especially for a Server since it is the sink for all reports. The conventional way to reduce bandwidth consumption by soft-state refreshes is slowing down the sending rate of refreshes. But in the case of huge data size, the interval between two consecutive refresh messages will be too big to be of any practical use. For example, when a packet is lost, the Server has to wait a long time before receiving the second transmission of the packet. Another problem caused by the data size is that Tracers have to periodically measure a huge number of distances, which can be a lot of workload.

The way we solve this problem is to differentiate data items based on their usefulness in distance estimation. Instead of maintaining a full distance map with all possible VLs, Servers calculate a t-spanner[3] from the full distance map and use the t-spanner in distance estimation. This t-spanner has much fewer VLs, but is still able to provide distance estimates without losing performance significantly. VLs in the t-spanner are called *active* VLs, while others are called *backup* VLs. Servers send feedback to Tracers informing them which VLs are active. Tracers will measure and report active VLs much more frequently than backup VLs. Therefore even some backup VL information become stale or inaccurate, the overall performance of distance estimation keeps at roughly the same level. Fig. 3 shows the overview of DIP's operation.

3. Protocol Design

3.1. Distance Map

The Number of APs and Tracers Before diving into details of protocol design, we would like to get an idea of the distance map's scale, namely the number of APs in the Internet and the number of Tracers needed by IDMaps.

First of all, the number of CIDR blocks is around 100K and growing. However, CIDR blocks are aggregated according to routing policies, not network distances as defined in IDMaps. The same CIDR block may have multiple APs. Therefore there are probably several times as many distinct APs as there are CIDR blocks. We conducted an Internet experiment in order to verify this and to get a more accurate estimation. We first obtain about 2 million web site names by searching English words in major search engines. After resolving names to IP addresses and cleaning them up, we end up with about 800K unique and reachable IP addresses as our IP address pool. Compared with BGP tables, our IP address pool covers about one third of the BGP address prefixes. Therefore, we assume it covers one third of the entire Internet, and the number of APs on the Internet will be roughly three times the number of APs that exist in our IP address pool.

In order to find the number of APs in our address pool, we first measure the round trip time (rtt) to all IP addresses in the pool and then group addresses that share the same prefix and have similar rtt into one AP. For this purpose we use the following algorithm. At first, we assume that all the addresses in the pool belong to the same AP. If the distance between any two addresses differs more than an error tolerance we split the AP in two halves. We recursively repeat the same procedure for every AP until all the addresses of each AP have almost the same distance (Fig. 4). We did the measurement from three different locations in USA[4] and the

[3] A t-spanner of a graph is a subgraph in which the distance between any pair of nodes is at most t times of the distance in the original graph.

[4] Though the experiment was carried out from only three sites, we feel the accuracy is adequate for our purpose of estimating the number of APs to assist the protocol design. We plan to expand the experiment to other locations as more tracers are deployed.

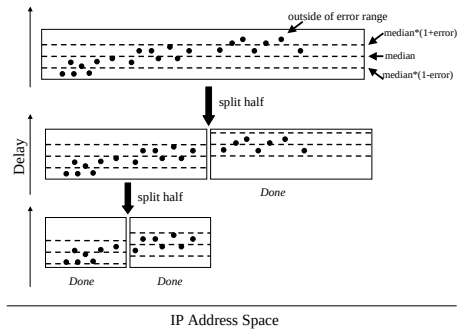


Figure 4. Simple top-down AP discovery

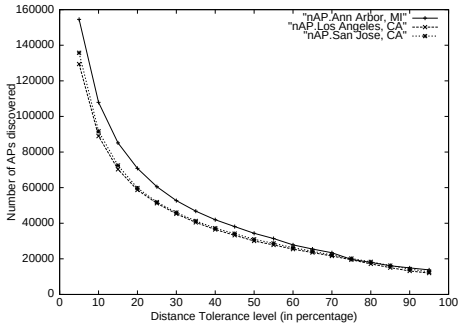


Figure 5. Number of APs discovered

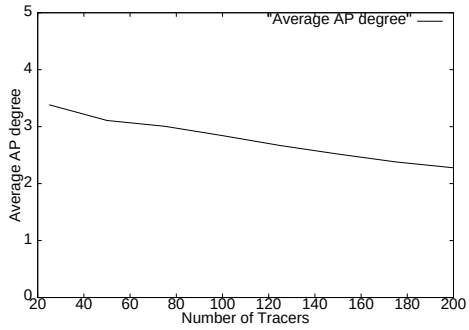


Figure 6. AP degree v.s. number of tracers

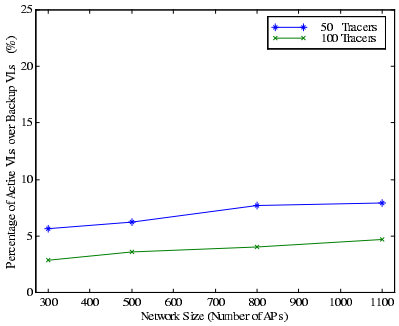


Figure 7. Active VLs v.s. backup VLs

results are shown in Fig. 5. If we conservatively take the error tolerance level as 10%, the number of APs is between 100K and 160K. Considering the address pool only covers about one third of the entire Internet, we estimate that there are about 300K to 500K APs on the current Internet.

We also did simulations to evaluate how many Tracers are needed by IDMaps in order to provide satisfactory distance estimation. The major conclusion is that increasing the number of Tracers will improve the performance, but with diminishing gain. Only 0.2% of all nodes serving as Tracers can already provide correct estimates with very high probability. Being conservative, we assume the number of Tracers IDMaps will deploy is 1% of the number of APs, which gives 3000 to 5000 Tracers. Detail discussions and simulation results are available in [3] and [7].

t-spanner Although the distance map can have as many as $\frac{1}{2}N * (N-1) + N * P$ VLs, not all VLs are equally important in distance estimation. For example, among three nodes *A*, *B* and *C*, if $d_{AB} + d_{BC} \cong d_{AC}$, then the *AC* virtual link can be well approximated by the sum of *AB* and *BC* virtual links. In [3], Francis *et al.* exploited this property by applying a t-spanner algorithm to the full-mesh of Tracer-Tracer VLs. The result shows that for t = 2, there is no perceptible performance degradation for applications like closest server selection, and the number of necessary Tracer-Tracer VLs are reduced by more than 10 times in a 1000-node topology with 100 Tracers.

What left unsolved in [3] is how many Tracer-AP VLs should be maintained, although it demonstrated that having 2 or 3 Tracers per AP performs better than having only one Tracer per AP. We solve this problem by extending the t-spanner algorithm to Tracer-AP VLs as well [7]. Now the entire distance map is reduced to a 2-spanner, and the distance estimation is done by calculating shortest path between two APs. Although one AP can have as many as *N* Tracer-AP VLs, our simulation (Fig. 6) shows that in the 2-spanner, the number of Tracer-AP VLs per AP, i.e., the AP degree, is very small, and it decreases as more Tracers are deployed.

Using a 2-spanner instead of the full distance map differentiates all VLs into two types: active VLs which comprise the 2-spanner, and backup VLs otherwise. In order to estimate the active to backup VL ratio, we run the 2-spanner algorithm on an Internet-like topology [14] of 5000 nodes, from which we randomly select APs and Tracers. The number of active VLs is much small compared with the number of backup VLs, as shown by Fig. 7. This makes it possible for DIP to allocate more resources to carry a small amount of important information, rather than treat every data item equally. We also designed an incremental t-spanner algorithm [6] to reduce the computational load.

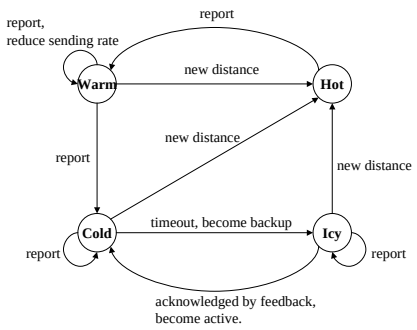


Figure 8. State transitions for virtual links

3.2. Tracer Report

Tracers report their distance measurement results to all the Servers by multicasting to the Server group Gs. The report is sent by best-effort instead of reliable delivery, which is expensive and unnecessary. Reliable delivery requires handling of NACK or ACK from multiple IDMaps Servers, which can be quite complex as reliable multicast protocols have shown. Since the report is sent periodically, a loss can be recovered by a later arrived report. Furthermore, when there is a new measurement result for a VL, whether the old information is delivered becomes irrelevant.

As discussed before, using the same sending rate for every VL does not work well when data size is very large. In order to scale to the data size, we need to take application requirements into account to allocate the bandwidth more efficiently. In DIP, Tracers categorize VLs into four types and use different sending rate for each type (Fig. 8). If a VL is a new VL or its distance has just changed significantly (e.g. over 50%) , it becomes *hot* and will be reported by a very fast rate R_{hot}. Then the VL enters the *warm* state as its sending rate gradually slows down, until a slow rate R_{cold} is reached and the VL becomes *cold*. Active VLs stay in *cold* state and the sending rate is fixed at R_{cold}. Backup VLs will go further to become *icy* where a much slower rate R_{icy} is in use. The transition from *icy* to *cold* is triggered and confirmed by Server's periodic feedbacks indicating which VL is active. If a VL has not been acknowledged by feedbacks for some time, it will become *icy* again. At any state, if the measurement reveals significant change in a VL's distance, the VL turns *hot*.

During *warm* state, the sending rate is exponentially reduced from R_{hot} to R_{cold}, cut in half after every transmission. Compared with directly jumping from *hot* to *cold*, this mechanism sends a few packets in fast rate at the beginning, so that the newly updated information is likely to overcome potential packet loss quickly. *Hot* and *warm* are transient states. Most of the time VLs are either *cold* or *icy*. To ensure quicker transmission of active VLs without starving backup VLs, Tracers adjust R_{cold} and R_{icy} so that bandwidth is allocated between them proportionally. In the current design, we tentatively set the bandwidth for backup VLs to be one third of the bandwidth for active VLs. Since active VL is only a small portion of all VLs, the number of backup VLs grows much faster than active VLs when the number of Tracers and APs increases. As a result, R_{icy} will become much lower. However this does not affect the system performance perceptibly since backup VLs are not directly used in distance estimation.

Although any significant change in distances will trigger fast transmission, we still need to send refresh message of active VLs and backup VLs in relatively slow rates (R_{cold} and R_{icy} respectively) periodically. First, it ensures the system's robustness by helping the Server overcome unforeseen faults. Second, for changes that are not qualified as significant changes, they will be conveyed to Servers via periodic refreshes. Though they may not make big difference in the distance map, such information is still useful in refining the distance estimation. Third, for a newly deployed Server, there will be a predictable time period before it collects most VLs, especially active VLs, in order to build the distance map and start providing estimation service.

3.3. Server Feedback

Positive Feedback A Tracer only measures distances between itself and other places and has no global view of the Internet distance map. Therefore it cannot decide on its own which VL is active, which backup. Such information is only available from Servers who are able to calculate t-spanner of the entire distance map.

Servers unicast *positive* feedbacks to Tracers telling them which VL is active, instead of *negative* feedbacks telling them which VL is backup. Positive feedback consumes less bandwidth than negative feedback since active VLs are much less than backup VLs. When the network is congested, lost negative feedbacks will trigger more report traffic since Tracers falsely think there are more active VLs. But lost positive feedback will just reduce the report traffic. Therefore using positive feedback is more scalable and robust.

Since Servers act independently in sending feedbacks, each Tracer will receive feedbacks from all Servers causing feedback implosion. Therefore we need an efficient feedback suppression mechanism to reduce duplicate feedbacks.

Feedback Suppression DIP uses a novel algorithm for feedback suppression. The goal is to elect a small council of Servers to send feedbacks to a Tracer. The council size should be appropriate to make sure the number of feedback that a Tracer receives is within the range of $[L \ldots U]$ and close to a constant c. The constant c is usually chosen

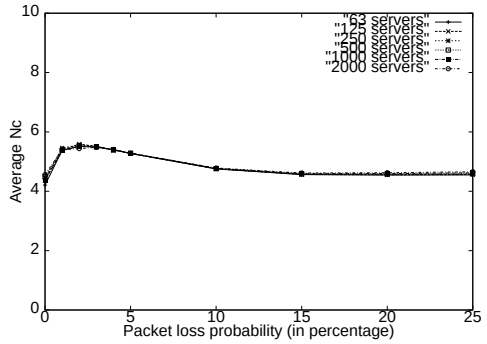


Figure 9. Nc vs. packet loss probability

as a small number greater than 1 in order to offset possible packet loss. The election of the council is done in each report-feedback round.

A simple way to elect the council is to let every Server hold a probability p and decide whether to join the council according to p. However, p is hard to choose because it depends on the total number of Servers and the packet loss probability in the network. The idea is to let Tracers tell Servers the result of last round election, and Servers adjust p accordingly to let the Tracers receive c feedback messages in the next round.

A Tracer includes the number of feedback it received (N_c) in last round with every report of active VLs. Each Server uses a flag s to remember whether itself is a council member ($s = 1$) or not ($s = 0$). A Server adjusts its probability of being a council member as follows:

$$p = \begin{cases} \frac{c}{N_c} & if\ N_c > U\ and\ s = 1 \\ 1 & if\ N_c \leq U\ and\ s = 1 \\ p_m & if\ N_c < L\ and\ s = 0 \end{cases}$$

Basically, if N_c is within the range [L...U], the council will remain unchanged; If there are too many feedbacks ($N_c >$ U), current council members will opt out by probability ($1 - \frac{c}{N_c}$); If there are too few feedbacks ($N_c <$ L), non-council Servers will join the council by a heuristic probability p_m. Non-council members maintain their p_m in the following way:

$$p_m = \begin{cases} p_m/2 & if\ N_c > U\ and\ s = 0 \\ p_m * 2 & if\ N_c < L\ and\ s = 0 \end{cases}$$

Initially p_m is set to $\frac{1}{2}$, and is always limited within [0...1]. After several rounds of adjustment, p_m will adapt to the number of non-council members. It makes sure that every time the council needs to increase, there will be only a limited number of new members joining instead of everyone.

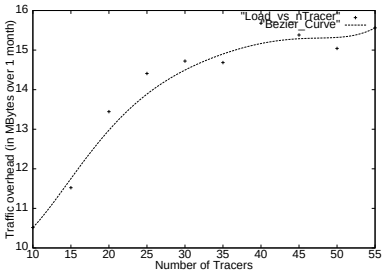
Compared with other feedback control mechanisms such as in [1], the council algorithm can elect the council with desired size quickly, normally in 1-3 rounds. The council remains the same most of the time, so the overhead of periodic re-election is saved. More prominently, the council algorithm adapts well to network conditions (e.g., packet loss) and scales independently of the number of Servers. Fig. 9 is a typical simulation result with L = 3, $c = 4.6$, and U = 6. It shows clearly that N_c remains close to c as the number of Servers varies between 63 and 2000, and the packet loss probability varies between 0% and 25%. When packet loss increases, more Servers will join the council so that the number of feedback received by the Tracer remains the same. Since the increment of council size is at least 1 every time, it may over-compensate the effect caused by packet loss when loss probability is small. This explains the initial increase of N_c in Fig. 9.

4. Performance Evaluation

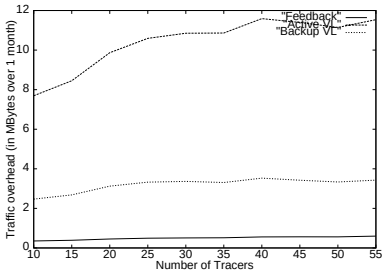
In this section, we use simulation to evaluate the overall scalability of DIP in terms of traffic load it introduces into the network. Our purpose is not to show the exact number of traffic volume, but to show the trend as some performance factors change. In particular, we try to find out how the traffic load changes as the system grows, what the contribution of different types of messages is, and how link failure affects the overall traffic load.

4.1. Simulation Setup

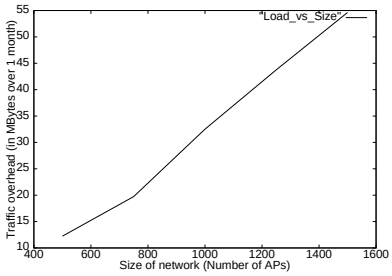
We use Waxman [15] topologies with parameters set to $\alpha = 1$ and $\beta = 0.1$ to model the network. These topologies are mesh-like with rich connectivity. When we simulate network changes by bringing a link down, there will be changes in network distances instead of partition of the entire topology. Most simulations are run on a 500-node topology with mean node degree of 5.37. Each node in the topology is treated as an AP. Tracers are placed into the network randomly. We tentatively set the rate of advertising active VLs, R_{cold}, to once every 2 hours, and restrict the bandwidth consumed by backup VLs to be one third of the bandwidth consumed by active VLs. Servers keep an active VL for at most 2.5*R_{cold} without receiving any refresh from the Tracer. In real implementation, these parameters should be determined by available bandwidth and the volume of distance information. The simulation runs for 1 month (43200 minutes) of simulated time. Network changes are simulated by link failure and recovery using a simple on-off model. The time that a link is down is exponentially distributed, as is the time that a link is up. The average length of downtime T_{off} is 4 hours, the link failure probability p_f is between 0% to 1.28%, and the average length of uptime is $T_{on} = \frac{(1-p_f)*T_{off}}{p_f}$. As a link goes up and down once in a while, the network path between two



(a) Traffic load v.s. number of Tracers



(b) Different types of traffic load



(c) Traffic load v.s. number of APs

Figure 10. Simulation Results

nodes may change, which in consequence may change some network distances, so that we can observe its effects on DIP.

4.2. Simulation Results

The main goal of our simulation is to compute the traffic load injected into the network due to DIP messages. Traffic load in our simulation is defined as the average number of bytes transmitted over one physical link, i.e., total traffic volume over the number of physical links. For multicast packets, we assume they traverse every link in the topology, to simplify computation.

Fig. 10(a) shows that the traffic load increases as the number of Tracers increases. When the number of Tracer is large, traffic load grows much slower than linearly. Fig. 10(b) shows the contribution from different types of messages. As designed, the traffic load of advertising active VLs is about three times as that of backup VLs. They grow sub-linearly because the number of active VLs grows sub-linearly with the number of Tracers. Feedback traffic only contributes a small portion of total traffic load. Because Servers use feedback to tell Tracers which VL is active, and only a small number of Servers are elected to send feedback to a particular Tracer, the feedback traffic is about proportional to the number of active VLs and is independent of the number of Servers deployed. All types of traffic grow sub-linearly, which makes the entire system scale well with the number of Tracers.

To see how traffic load changes with the number of APs, we run several simulations with different network sizes. The numbers of Tracers and IDMaps Servers are fixed (25 Tracers and 5 IDMaps Servers). The link failure rate is 0.01% and topologies are created with the same Waxman parameters. Fig. 10(c) shows that the traffic load increases linearly when the size of the topology changes from 500 to 1500. Given the number of APs on the Internet is about 500-700 thousands, extrapolation of Fig. 10(c) gives an estimated average traffic of 43-25Kbps when DIP is deployed widely.

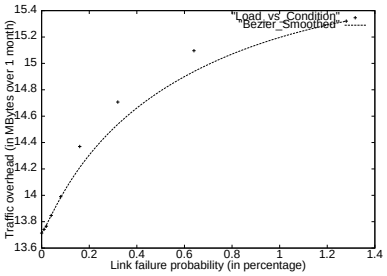


Figure 11. Traffic load vs. link failure

To show how the traffic overhead changes under different network conditions, we simulate DIP in a 500-node topology with 25 Tracers and 5 IDMaps Servers and apply the on-off model of link failure. Fig. 11 shows that the traffic load grows as the link failure probability increases, but its pace slows down as the network becomes less stable. There are two factors that affect the traffic load in this simulation. One is how frequently the network changes. When links go up and down, the distance between two nodes are likely to change which triggers Tracers to advertise new distances at a fast rate. In DIP's term, more VLs become *hot*. Therefore, more network changes cause more traffic load. On the other hand, when link failure probability is high, some links will be down for a long time. Statistically, there are fewer links in the network topology. This results in less links in the t-spanner because more virtual links will be able to share the same physical link. Therefore, although the total number of VLs remains the same, there are less active VLs in this statistically simplified topology, and traffic load is reduced. As shown in Fig. 11, when link failure probability is small, the first factor dominates and traffic load increases almost linearly. When link failure probability is high, the second factor gradually offsets the first factor, thus the curve goes

flatter. This simulation demonstrates both DIP's adaptability to network changes and the effectiveness of information differentiation.

5. Related Work

Soft-state approach has been adopted by many Internet protocols, including RSVP [18], SRM [2], SAP [5] and so forth. SAP is similar to DIP in that senders periodically advertise information to a well-known multicast group which receivers passively listen. SAP controls its bandwidth consumption by slowing down the advertisement when the number of sessions increases. State compression [11] is a technique that compresses RSVP state information to reduce bandwidth consumption of refresh messages. It is orthogonal to our approach of information differentiation. It can be applied to DIP to further reduce transmission overhead, but cannot help reduce Tracer's measurement workload.

The idea of sending important data faster appeared in [8], in which RSVP triggered messages are sent out quickly, then exponentially slowed down to a fixed refresh rate after an acknowledgment has arrived from the receiver. Sharma *et al.* [10] presents an adaptive algorithm for senders to adjust refresh rate and for receivers to set timeout timer dynamically. SRM [2] proposes to organize data into a hierarchical namespace and periodically multicast this namespace for the purpose of detecting packet loss and recovering the loss. Raman and McCanne [9] summarize many of these techniques and propose a formal model to analyze the performance tradeoff in soft-state protocols. These work focus on reliable data delivery, while DIP focuses on how to disseminate large-scale data set.

6. Summary

Designing a large-scale data dissemination protocol for IDMaps is challenging in that the protocol has to deal with both the dynamic operational environment and the huge data size. A hard-state design does not need periodic refreshment, but requires reliable delivery and complex error handling. Basic soft-state design handles various errors by simple periodic refreshment, but the traffic load will be prohibitive as the data size is huge. DIP takes the soft-state approach and incorporates several techniques such as staged timer, positive feedback, feedback suppression etc. under the guidance of information differentiation. We demonstrate that by taking application's requirements into account to allocate resources more efficiently, a soft-state data dissemination protocol can be made scalable to huge data size. We believe this principle of information differentiation can be applied to other soft-state protocols as well to achieve better scalability.

References

[1] J. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the Internet. In *Proc. of ACM SIGCOMM*, pages 58–67, 1994.

[2] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *ACM/IEEE Transactions on Networking*, 5(6):784–803, 1997.

[3] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global internet host distance estimation service. *ACM/IEEE Transactions on Networking*, Oct. 2001.

[4] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. An architecture for a global internet host distance estimation service. In *Proc. of IEEE INFOCOM 1999*, Mar. 1999.

[5] M. Handley, C. Perkins, and E. Whelan. Session announcement protocol. RFC 2974, IETF, Oct. 2000.

[6] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of Internet instrumentation. In *Proc. of IEEE INFOCOM 2000*, Mar. 2000.

[7] Y. Jin. *Distance Information Protocl (DIP) for Internet Distance Map Service (IDMaps)*. PhD thesis, University of California, Los Angeles, June 2001.

[8] P. Pan and H. Schulzrinne. Staged refresh timers for RSVP. In *Proc. of IEEE GLOBECOM*, pages 3–8, November 1997.

[9] S. Raman and S. McCanne. A model, analysis, and protocol framework for soft state-based communication. In *Proc. of ACM SIGCOMM*, pages 15–25, 1999.

[10] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable timers for soft state protocols. In *Proc. of IEEE INFOCOM*, pages 222–9, April 1997.

[11] L. Wang, A. Terzis, and L. Zhang. A new proposal for RSVP refreshes. In *Proc. of the Int'l Conf. on Network Protocols (ICNP)*, pages 163–72, November 1999.

[12] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proc. of the Int'l Workshop on Networked Group Communication (NGC)*, Oct. 2002.

[13] Z. Wang, A. Zeitoun, and S. Jamin. Challenges and lessons learned in measuring path RTT for proximity-based applications. submitted for publication, 2002.

[14] J. Winich and S. Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.

[15] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM*, Mar. 1996.

[16] B. Zhang, S. Jamin, and L. Zhang. Host Multicast: a framework for delivering multicast to end users. In *Proc. of IEEE INFOCOM*, June 2002.

[17] B. Zhang, S. Jamin, and L. Zhang. Universal IP multicast delivery. In *Proc. of the Int'l Workshop on Networked Group Communication (NGC)*, Oct. 2002.

[18] L. Zhang, S. Deering, D. Estrin, S. Schenker, and D. Zappala. RSVP: A new resource ReSerVation protocol. *IEEE Network Magazine*, 7(5):8–18, Sept. 1993.