New Models and Algorithms for Programmable Networks

Danny Raz Yuval Shavitt
{raz, shavitt}@research.bell-labs.com
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733-3030

Abstract—**In todays IP networks most of the network control and management tasks are performed at the end points. As a result, many important network functions cannot be optimized due to lack of sufficient support from the network. The growing need for quality guaranteed services brought on suggestions to add more computational power to the network elements.**

This paper studies the algorithmic power of networks whose routers are capable of performing complex tasks. It presents a new model that captures the hop-by-hop datagram forwarding mechanism deployed in todays IP networks, as well as the ability to perform complex computations in network elements as proposed in the active networks paradigm. Using our framework, we present and analyze distributed algorithms for basic problems that arise in the control and management of IP networks. These problems include: route discovery, message dissemination, topology discovery, and bottleneck detection.

Our results prove that, although adding computation power to the routers increases the message delay, it shortens the completion time for many tasks. The suggested model can be used to evaluate the contribution of added features to a router, and allows the formal comparison of different proposed architectures.

I. Introduction

The design of todays IP networks is based on the end-to-end argument, that calls for the internal network elements to concentrate on the simple forwarding function, leaving the network control and management to the end points. As a result, many important network functions cannot be optimized due to lack of sufficient support from the network. To alleviate this problem, in many proposed architectures the network elements are enhanced with more processing abilities. Active networks [17], [13] are an extreme example of the break away from the end-to-end argument [1]. In these networks even the forwarding and routing functions may be done in software. This paper studies the algorithmic power of networks whose elements, namely routers, are capable of performing complex tasks. We use the term active networks in this paper to relate to any such network.

The main contributions of this paper are as follows. First, it presents a new model that captures the hop-by-hop datagram forwarding mechanism deployed in todays IP networks, as well as the ability to perform complex computations in network elements as proposed in the active networks paradigm. Then, we use this framework to present and analyze distributed algorithms for basic problems that arise in the control and management of IP networks. Some of these algorithms were implemented in an active network test bed [14]. These problems include: route discovery, message dissemination, topology discovery, and bottleneck detection. In particular we show that for many needed tasks, adding computation power to the routers reduces the overall resource utilization. The suggested model can also be used to evaluate the contribution of added features to a router, and allows the formal comparison of different proposed architectures.

To the best of our knowledge, there is no algorithmic framework to evaluate and compare algorithmic solutions for active networks. The existing models for asynchronous distributed computing do not capture the way information and computation interact in the active networks paradigm. In particular, the ability to perform complex computations in the network as part of the packet handling, is not captured by traditional models. We seek to define a model that can capture the new functionality enabled by active networks and the penalty of the additionally introduced end-to-end delay. The existing models assume that a distributed algorithm is executed in phases (not necessarily synchronous) of computation and transmission along a point-to-point link. In the asynchronous model, usually used for network protocol analysis, the delays are finite but unbounded. I.e, the correctness of an algorithm cannot rely on a specific bound on the delay. However, for the purpose of time complexity, each of the delays is bounded by some constant. Even in cases where separate bounds are used [9, Part IIB], the bounds on the algorithm time complexity

Danny Raz is also with the Dept. of Computer Science, Technion — Israel Institute of Technology, Haifa, Israel. danny@cs.technion.ac.il
Yuval Shavitt is also with the Dept. of Electrical Engineering - Systems, Tel-Aviv University, Tel-Aviv 69978, Israel. shavitt@eng.tau.ac.il

are always a function of the sum of these bounds. This is due to the fact that the model assumes that each transmission of a message triggers a computation in the receiving node, and communication is only allowed with immediate neighbors.

The problem with the above model is that it fails to capture the way distributed algorithms are implemented in many modern network designs, e.g., high-speed networks [3] and active networks [14]. Cidon et al. [4] were the first to address this problem in the context of high speed networks. They separated the delay associated with transmission and the delay associated with computation, and allowed messages to cut-through a node with no computation delay penalty. However, their model is not suitable for IP-like networks since it assumes a source routing mechanism rather than hop-by-hop routing. In addition, the bound on the computation performed at a node (NCU in [4]) is constant and does not depend on the complexity of the computation performed. This is suitable when all the computations performed in the network are simple in nature, but may not be appropriate when arbitrary complex computations may be used.

We suggest here an algorithmic model in which active networks algorithmic solutions can be evaluated and compared against each other and against traditional solutions that only use computations at the network edge devices. The model captures the delay inflicted by the computations performed in the network. The main strength of the model is its ability to capture the way the delay of a packet passing through a router changes depending whether the packet is handled by some software intensive process or whether it cuts-through the router using only traditional IP forwarding functions. Our model can serve not only to evaluate algorithms but also to compare the strength and expressiveness of different software execution environments in the routers. Using this algorithmic model, we can derive lower bounds on the capabilities of different execution environments to perform tasks. It can also be used to evaluate the contribution of new features to an existing architecture.

Our model captures an active IP network, i.e., the active nodes are part of a network that employs IPv4 routing. As in most currently suggested models [14], [5], packets arriving at a node (router) can be treated in two distinct ways: processed by some software centric process that is slow by nature, or forwarded via a 'fast track' (called cutthrough in [5]) that adds a negligible delay to the IP forwarding delay. The fast track, usually, involves some type of filtering that may be done in hardware or in software as part of the usual IP filtering. The slow track may be done in a physically different machine [14], or in an isolated environment and thus its delay may contain a large constant element associated with the activation of the relevant software unit, and a linear (in the message size) element associated with reading and performing calculations on the data contained in the message.

The model is then used to develop and analyze algorithmic solutions for several basic problems that arise in network control and management. Some of these problems where discussed in [14] where we described an active network implementation and its usage for network management. We also suggested there some basic algorithmic solutions, and discussed their asymptotic complexity. Similar observations were later made by Chae et al. [2]. However, without the theoretical framework developed in this paper, the analysis of the algorithms presented in [14] and in [2] is very preliminary, e.g., the delay in the slow track is not considered. Using the model we developed here, enables us to develop superior algorithms for these basic problems. The performance measures we use are time complexity and message complexity. A good algorithmic solution may aim at one of these measures or at some best tradeoff of the measures.

We first consider computations along a path. The most basic problem here is to identify the ids of the nodes along a route between two end points.[1] We analyze several algorithms for this problem, discuss the trade off between the different complexity measures, and finally present an algorithm that is optimal (up to a $\log$ factor), both in terms of time and message complexity. Another closely related problem is to compute any function along the route. Bottleneck detection is studied as an illustrative example for this class of problems. In this case we seek to find the most congested link along a path. A second class of problems deals with the optimal delivery of a message to a (possibly large) group of nodes. The assumption here is that the group is an ad-hoc collection of destinations targeted by this one message and not some long lived multicast group. The algorithms we suggest for the above problems are used as building blocks for solving the basic networking problems of topology and routing discovery. We also consider a sub-class of globally insensitive functions [4], that can be computed using only constant memory, such as counting the network nodes, searching for a color printer, or computing the network average load.

The rest of the paper is organized as follows. The next section describes the formal model. In Section III we discuss the problems of calculating functions along a path. Section IV examines the message dissemination problem, Section V deals with topology discovery, and Section VI discusses computation of global functions. We discuss the results and further work in Section VII.

[1] This is a special case of functions that cannot be computed with less than linear memory, such as the mode (most frequent element) of a vector.

II. MODEL

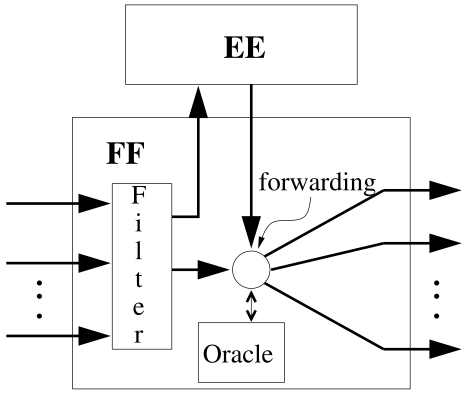

Fig. 1. A node structure

The network is represented by a connected graph $G = (V, E)$, where V is the set of nodes and E is the set of bidirectional communication links. Each node (see Figure 1) consists of a fast forwarding unit (FF), e.g., a legacy IP router, which is attached to the communication links, and an execution environment (EE). A packet suffers forwarding delay at every hop, which consists of transmission delay and queueing delay. In some cases, the packet is handled by the EE, in which case, it will suffer an additional processing delay. The communication links preserve the order of packets handed to them for transmission (FIFO order), the EE serves the programs in a FIFO order, as well.

A packet header is comprised of a source address, a destination address, and an application identifier. The FF matches each arriving packet header against a set of filters. If a match is found the packet is diverted to the EE, otherwise the packet is forwarded only according to the destination addresses. More formally, at each node i, there exists an oracle forwarding function $F_i : V \to \mathcal{N}_i$ which for every destination $v \in V$ returns the next hop on the route from the group of i's neighboring nodes, $\mathcal{N}_i$. Motivated by distance vector protocols [10], such as RIP [6], the forwarding function and the list of immediate neighbors is the only information available at a node. We do not assume any restrictions on the routing function. In particular a route to a parent may be different from a route to a child, i.e., the route a→b→c→d→e may co-exist with the route a→x→y→z→d. However, for the purpose of complexity analysis, we assume that the underlying routing algorithm is a shortest path algorithm. All the presented algorithms work correctly without this assumption. We denote the network diameter as defined by the underlying routing by D.

A. Motivation

The model captures the structure of a router with IP-like forwarding that is performed by some fast forwarding hardware, and general execution environment that is capable of performing complex computations. It can be easily used to analyze algorithms in IP networks, active networks with IP as an optional execution environment [5], or IP networks enhanced with programmable management capabilities [14]. An IP packet has many fields some of which were defined to aid in specific tasks not directly related to the IP function. Our aim is to capture the basic behavior of the protocol, the one that is likely to be included in any datagram networks forwarding protocol.

The EE is an execution environment in which programs that are stored in the packet or referenced by it can be executed. Although we use the same term as in [5], the EE here is an abstraction of the entire active environment defined in [5] that may include some of the functionalities of the NodeOS, and one or more EEs. In the most general case, the programs can perform any computation both on the packet's data, and on local data (e.g., soft state left by other packets, or local topology information). Therefore the delay of a packet through the EE may be a function of the data size. This function reflects the complexity of the algorithms in the program.

B. Performance measures

We follow the standard model for asynchronous communication networks [9]. For the correctness of the algorithms all delays associated with a packet are assumed to be finite but unbounded. However, for the purpose of time complexity analysis, delay is upper bounded. In our case, we assume that the delay imposed by the FF is upper bounded by a constant C (We preserve the notation of Cidon et al. [4]). The process of diverting a packet to the EE imposes a constant delay, as a result of the traversing the communication protocol stack. The execution of the program at the EE introduces an additional delay which is a function of the algorithm in use and the size of data it is working on, k. The overall EE delay is, thus, bounded by $P(k)$, a function of the executed program complexity and the constant cost. Thus, a packet that passes only through the fast track suffers a delay of C time units, while a packet that passes through the EE is delayed by $C + P(k)$ time units.

The function $P(k)$ depends on the computation that is performed at the EE. However, since in order to send a message to the EE a node has to copy it, the function $P(k)$ is at least linear. For the rest of this paper we assume $P(k) = P_c + kP_l$, where k is the packet length, and in many cases, P_c is negligible. Thus, for simplicity we use $P(k) = kP$, where P is a constant. Investigating other processing delay functions is an interesting future research topic. In most cases, since the FF is implemented by specialized hardware while the EE is a general purpose processor, $C < P(1)$. For a discussion about some measured values of P_c and P_l see section III-A.

We neglect the cost of transmitting the programs themselves. We believe, that most active programs will be used extensively for a long time period and thus be cached in the nodes. The cost of getting the program to the node cache can be amortized over all its many activations. In other cases, where small scripts are tailored for specific applications [7], [15] the cost of sending them is either negligible or can be introduced through the function $P(k)$.

We want to compare the algorithm performance both from the application point of view (time delay) and from the network point of view (efficient use of network resources). The following performance measures are, thus, of interest [4]: time complexity — measures the time that may elapse from a task starting point to its termination; and communication complexity — the number of hops traversed by packets to perform a given task. Other complexity measures such as bit complexity or memory usage are not discussed here.

III. Route Exploration

In our model, unlike the model of Cidon et al. [4], global routing information is *unavailable* at a node. Retrieving this information, such as the route between two nodes, is important both to user and network management applications. In IP networks, the route detection problem is addressed by the `traceroute` program, which is a hack that uses the TTL field in the IP header.

The problem can be stated as follows. A node v seeks to learn the ids of the nodes along the route from itself to another node u. Recall that in our model, v only knows the id of the next hop node along this route. In a naive implementation (naive), node v queries this node for the id of the second hop node. Then it iteratively queries the nodes along the route until it reaches the one leading to the destination. This method resembles the way the `traceroute` program works, but it does not use the TTL field which is not part of the model. The delay of the naive algorithm is comprised of n activations of an EE level program plus the network delay of $2i$ for $i = 1, 2, \ldots, n-1$ hops. The message complexity is given by $\sum_{i=1}^{n-1} 2i = O(n^2)$ (see Table I and Fig. 2).

Next we describe two simple algorithms, collect-en-route and report-en-route, (presented in [14]) that improve the above solution to the route exploration problem, and analyze their performance using our model. Following this discussion we turn to more sophisticated solutions that achieve near optimal performances.

In algorithm collect-en-route (see Fig. 2(C)) the source initiates a single packet that traverses the route and collects the host ids from each node. When the packet arrives at the destination node, it sends the data directly (using only the FF) back to the source. Clearly, the message complexity of this algorithm for a node at distance n is exactly $2n$.

The communication delay for this algorithm is $2nC$ since exactly one message traverses the route in each direction. The execution delay at a node at distance i is iP since the message length is increased by one unit each hop. The total delay is given by $2nC + \sum_{i=1}^{n} iP = 2nC + \frac{n(n+1)}{2}P$. Note that this algorithm is somewhat more sensitive to packet loss than the previous (and the following) one since no partial information is available at the source before the algorithm terminates. Furthermore, the time-out required to detect a message loss here is significantly larger than with the other algorithms presented here.

In algorithm report-en-route (see Fig. 2(B)) the source sends a request packet downstream the path. When a request packet arrives at a node, it sends the required information back to the source and forwards the request downstream to the next hop. This design minimizes the time of arrival of each part of the route information, while it compromises communication cost. The message complexity is clearly $O(n^2)$, while the processing complexity is $O(n)$.

The communication delay for this algorithm is $2nC$ since exactly one message traverses the route forwards till the destination, and this message is then sent back to the source. The execution delay in all the nodes is P since the message length is exactly one unit. The total delay is given by $n(2C + P)$.

Algorithm collect-en-route features a linear message complexity with a quadratic delay, while algorithm report-en-route features a linear completion delay with a quadratic message complexity. Combining these two algorithms we can achieve tradeoffs between these two measures. In particular, if both measures are equally important we may want to minimize their sum.

An algorithm that enables us to optimize the two measures combined works as follows (see Fig. 2(D)). First step is to obtain n, the length of the route between the two endpoints. This may be known from previous executions of the algorithm or can be obtained by an algorithm which is linear both in time and message complexity (see section III-C). Next we send a fixed size message to initiate collect-en-route in n/l segments each of length l. This can be done using a counter that is initialized to l at the beginning of every segment, and decreased by one at every intermediate node. Thus, the execution of collect-en-route in the segment i starts after at most $(i-1)(C+P)$ time units, and the overall time complexity is $(n-l)(C+P) + \sum_{i=1}^{l}(C+iP) = O(nC + (n+l^2)P)$.

The message complexity is $O(n)+\sum_{i=1}^{n/l}(l+il) = O(\frac{n^2}{l})$. Choosing $l = \sqrt{n}$ results in a linear time complexity while using $O(n\sqrt{n})$ messages. The requirement to balance the two measures (up to a constant factor of P), translates to $l^2 = \frac{n^2}{l}$ which gives $l = n^{2/3}$. For this l value both time and message complexity are $O(n^{4/3})$.

A different approach, however, is needed to reduce both the time and message complexity. The following algorithm, collect-rec, achieves an almost linear time and linear message complexity. The main idea is to partition the path between the source and the destination into two segments, to run the algorithm recursively on each segment, and then to send the information about the second segment route from the partition point to the source via the FF track. In order to do so on the segment (i, j), in each recursive step one needs to find the id of the partition point, k, and to notify this node, k that it has to perform the algorithm on the segment (k, j) and report to i. In addition, i has to know that it is collecting data only until this partition point, k, and it should get the rest of the information via the fast track. The partition can be done, naively, in two passes. First we find the segment length. Then sending the segment length and a counter in the slow track allows k to identify itself as the partition node. This will result, for a segment of length n, in a $2nP+3nC$ time complexity, and $3n$ message complexity. Sending the data from the partition point using the FF track, requires $n(P+C)$ time, and adds at most n to the message complexity. We thus get for the time complexity

$$TC(n) \leq TC(n/2) + 4n(P+C),$$

and for the message complexity

$$MC(n) \leq 2MC(n/2) + 4n.$$

By solving these equations we can prove the following theorem.

Theorem III.1: Algorithm collect-rec solves the route detection problem with time complexity of $O(n)$, and message complexity of $O(n \log n)$.

□

A pseudo-code implementation of this algorithm is given in the appendix.

A. Numerical Example

In this section we use measurements made in active network prototypes to evaluate what the real values of the delay bounds C and $P(k)$ may be. One must take the numbers we quote here with great caution: they are taken from research prototype systems, and they reflect average delays rather than delay bounds. The quotation of these measurement serves two aims: first, the numbers support our basic motivation for the need of a new model with a distinction between the delays of the FF and the EE. In addition, we use it as an example of a real use of our general model to evaluate the performance of the algorithms described and analyzed in Section III for a real active network.

In PLANet, a system built at The University of Pennsylvania, Hicks et al. [8] measured the traversal delay through a software router. The measured delay through the router running in kernel mode was $C(k) = 36\mu\text{S} + k \cdot 0.13\mu\text{S}$. For PLAN programs the delay was $P(k) = 259\mu\text{S} + k \cdot 0.16\mu\text{S}$. The program they used for their measurement had a running time independent of the packet length which may explain the small increase in k's coefficient compared with the significant increase in the constant element.

In Bowman, a NodeOS built at The Georgia Inst. of Technology, Merugu et al. [11] measured packet processing delay through the NodeOS and found that $P(k) = 560\mu\text{S} + k \cdot 1.1\mu\text{S}$ (see [11, Section IV-A.1]). There is no direct delay measurement through their software router, but the throughput data shows that the gap is between

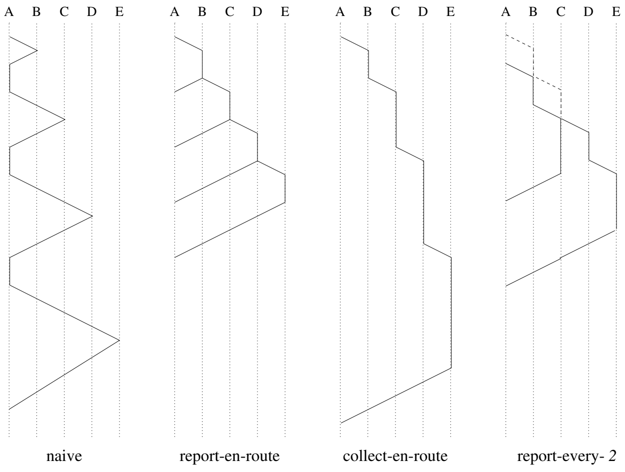

Fig. 2. Example of route exploration executions on a four hop route: (A) naive; (B) report-en-route; (C) collect-en-route; and (D) report-every-l.

algorithm name	time complexity	message complexity
naive	$O(nP + n^2C)$	$O(n^2)$
collect-en-route	$O(n^2P + nC)$	$O(n)$
report-en-route	$O(nP + nC)$	$O(n^2)$
report-every-l	$O((n + l^2)P + nC)$	$O(n^2/l)$
collect-rec	$O(nP + nC)$	$O(n \log n)$

TABLE I
SUMMARY OF ROUTE EXPLORATION ALGORITHMS.

200% for small 200 byte packets to 5% for packets over 1200 bytes. Here, too, the processing of the packet was independent of the packet length.

In PAN, a capsule-based active node built at MIT, Nygren et al. [12] measured the delay through several configurations of an active node. When only IP forwarding was used, the delay measured for a 128 byte packet was $C(128) = 50\mu S$, and for 1500 byte packet $C(1500) = 130\mu S$. These numbers are very close to the ones measured in PLANet [8]. The active evaluation delay for 128 byte packets ranges from $70\mu S$ for kernel implementation using native code to close to $500\mu S$ for a Java implementation.

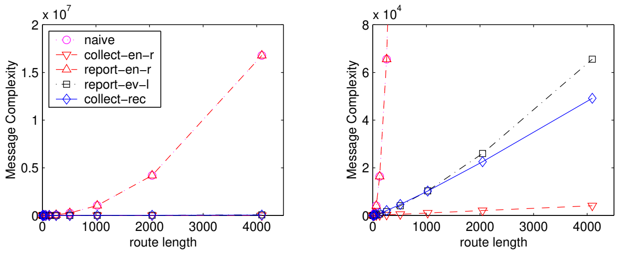

Fig. 3. The message complexity of the route exploration algorithms as a function of the route length.

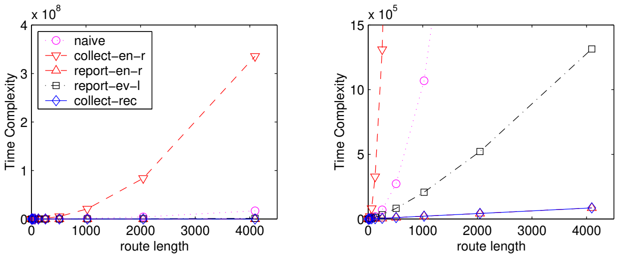

Fig. 4. The time complexity of the route exploration algorithms as a function of the route length for $C = 1$ and $P = 20k$.

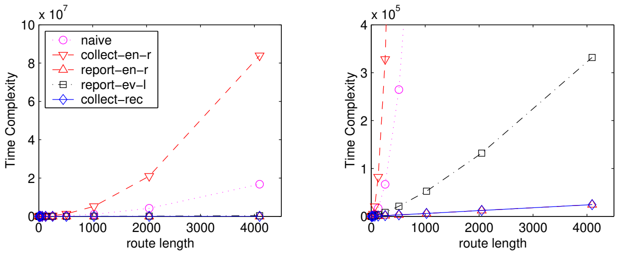

Fig. 5. The time complexity of the route exploration algorithms as a function of the route length for $C = 1$ and $P = 5k$.

Figures 3 – 5 depict the message and time complexities of the five route exploration algorithms described before as a function of the route length. We selected two P/C ratios, 5 and 20, based on the measurements reported above. Since the quadratic curves hide the differences between the sub-quadratic functions, the right graph of each of the figures is scaled to show the behavior of the latter. Figure 6 depicts the tradeoff between the time and message complexities. For each algorithm we plot how the time-message complexity operation point advance in the time-message complexity space. We plotted the values at the natural powers of two from 1 to 12. Note that the collect-rec curve is much shorter than the rest, as it represents a close to optimal trade-off.

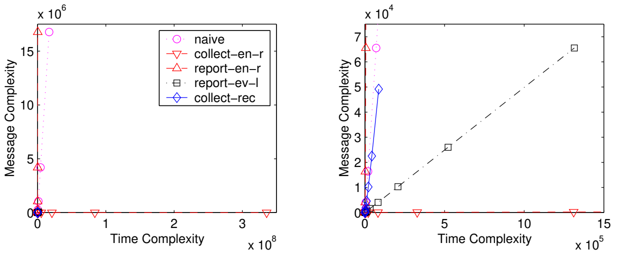

Fig. 6. The behavior of the route exploration algorithms in the message and time complexities domain as the route length changes when $C = 1$ and $P = 20k$.

B. Lower bounds

The message complexity lower bound for the route detection problem is $\Omega(n)$. The proof is trivial since one has to query at least $n - 1$ of the nodes on the route. To reach, say, a node in the middle of the route from either end, will cost $\Omega(n)$ messages. The processing complexity lower bound is $\Omega(n)$ since one has to query at least $n - 1$ of the nodes on the route.

A trivial time complexity lower bound for the route detection problem is $\Omega(P + 2nC)$, since a message should, at least, reach the destination (to be exact, one hop before the destination, since the destination id is known there) and then be sent back to the source. The propagation delay along an n hop route is $2nC$, and the delay caused by the application that turns the message around is P. Next we state a tight lower bound, the proof is given in the appendix.

Theorem III.2: The time complexity lower bound for the route detection problem is $\frac{1}{2}(n-1)(P + C)$.

Note that this lower bound holds for an arbitrary message length, and an arbitrary processing time at the EE.

C. Bottleneck detection

In many cases exploring the path between two nodes is just an intermediate step towards computing some function along this path. A typical example is bottleneck detection: we want to detect the most congested link along a path. For example, in QoS routing, the maximum available bandwidth in a path is an important criterion in selecting a path. One way to implement it is by using modified versions of any of the previous route exploration algorithms. In these versions, the algorithm collects, in addition to the nodes id, also its current load. The bottleneck is computed at the source based on the collected information. However, there is a better alternative. Consider the following (compute-en-route) algorithm: a packet containing the id of the most congested link so far and its load value is sent along the path. Each node updates it, if it is more congested, and forwards it towards the destination. The last node sends the report directly (using FF) to the initiator. Since the message has a fixed size along the path, unlike in the report-en-route case where it was linear, the time complexity here is only $O(n(P+C))$. The message complexity and the computation complexity are the same as in report-en-route.

Bottleneck detection is a special case of a one-path linear function (termed succinct functions). These functions, e.g., average, min, max, and modulo, can be computed with a single path on the input, requiring only constant amount of memory.

Theorem III.3: Every succinct function on a path can be computed with time complexity $\Theta(n(P+C))$ and a message complexity of $\Theta(n)$, and these bounds are tight.

Proof: The compute-en-route algorithm can be used to compute any succinct function along a path. Since the message size is constant the time complexity is linear. The message complexity is linear since at most one message traverses each link along the path in each direction. Notice that the lower bound we proved in the previous section holds for this case, thus the compute-en-route algorithm meets both the time and message complexities lower bounds to compute a succinct function along a path. ■

IV. Message Dissemination

In many applications there is a need to deliver a message to a group of nodes. In such cases, the group of nodes is defined ad-hoc for the purpose of a single message dissemination and it is not a long lasting group as in multicast applications. Since the group is defined by the recipient list of a single message, it is not efficient to form a multicast group or to invest in any other long term infrastructure. A trivial solution to disseminate the message to the group is to send a unicast message to each node. The time complexity of this solution to a recipient group of size m is bounded by $mP+DC$. The first term is due to the processing delay at the sender, while the second is the propagation delay using the FF infrastructure. The message complexity, however, may be mD.

We assume that a message is comprised of a header with a list of receivers, and a body which, for a large group of receivers, is much smaller than the header. Thus for the analysis purpose the message size is approximated by the size of the recipient list it carries.

Notice that the union of all the routes from the originator to the receivers is a DAG rooted at the originator. From this DAG we create a directed tree, termed *the dissemination tree*, by splitting every node with i incoming links into i nodes, recursively. The tree height, h_T, is bounded by the network diameter, D. The number of nodes in the dissemination tree, n_T, is bounded by mh_T since there are m recipients each at distance not longer than $h_T \leq D$. We denote d_T the maximum out-degree of a node in the dissemination tree.

Our solution is to partition the recipient list at the source according to the first hop on the path to each recipient. We continue this partitioning at every intermediate node until the message arrives at the tree leaves. This way, exactly one copy of the message traverses each link in the dissemination tree. For example, in a balanced binary tree with m leaves, our message complexity is $2m$, while the unicast solution has a message complexity of $m \log m$.

At each node, the processing time is linear in the recipient list length. Recall that this list contains only the addresses of the leaves in the subtree rooted at the node. The propagation delay is bounded by $h_T C$. The processing delay is bounded by $h_T mP$. Thus the time complexity is bounded by $h_T(C+mP)$. The actual delay depends on the tree structure. For example, in a balanced binary tree with m leaves, $h_T = \log m$. In addition, for this special case, the number of leaves for a node at distance $\log(m-i)$ is 2^i, and thus the processing delay is $O(mP)$ and the time complexity is $O(\log mC + mP)$. Note, that for this case we achieve a logarithmic improvement in the utilization by paying only a constant factor in delay.

V. Topology Discovery

A. Node id discovery

A node in the network wishes to find the network topology, i.e., the ids of all the nodes in the network and the links that connect them. One simple algorithm, report-direct, is to use Segall's PIF [16] algorithm, to request from all the nodes to report the id of their neighbors directly to it. This algorithm is the fastest possible (see below), but it is suboptimal in its use of bandwidth. A different algorithm, report-on-tree, uses the tree built by PIF to collect the topology information as follows. A node starts the algorithm by sending a PROBE message with its id to all its neighbors. Every node that receives a PROBE message for the first time, marks the neighbor it received the message from as its parent, and sends a PROBE message with its own id to all its neighbors but the parent. A node that received a message from all its neighbors sends a REPORT message to its parent. The REPORT message contains all the messages received by the node from its neighbors but the parent. This algorithm is similar to the connectivity test algorithms suggested by Segall [16]. The proof of the following theorem is left to the appendix.

Theorem V.1: Algorithm report-direct is the fastest possible for topology discovery.

The time complexity of algorithm report-direct is bounded by $(|V| + D - 1)C + (|V| - 1)P$. The forward phase of building the tree is bounded by $(|V| - 1)(C + P)$, while the backwards report is bounded by DC. Recall that D is the network diameter as defined by the underlying routing. The message complexity of the first stage is clearly $2|E|$ since a PIF message traverses each link once in each direction. The report messages travel up to D links, and hence, the overall message complexity is bounded by $2|E| + |V|D$.

An obvious bound on the time complexity of report-on-tree is $O(|V|^2 P + |V|C)$ since no node can receive more messages than its degree which is bounded by $|V| - 1$.

B. Routing discovery

Recall that in our model the only routing information at a node is the next hop to any destination. If a node wishes to know the global routing structure from itself to a group of nodes, or from a group of nodes to itself the topology information may not be sufficient.

This information may be obtained by using any of the algorithms described in section V-A, by requiring each node to send its full forwarding table. This may increase the time complexity by a factor of $|V|$. In this section we discuss better alternatives to address this problem.

First examine the case where a node wishes to learn the incoming route to itself. If it wished to learn the path from *every* node in the network to itself, we can use any of the algorithms of section V-A and ask each node to send its id and the id of its next hop neighbor on the route to the originator. This will only add a factor of 2 to the time complexity, without increasing the message complexity.

When the route from a subgroup of the nodes is of interest, the algorithm works in two phases. In the first phase a message is sent to the subgroup using the message distribution algorithm of section IV. In the second phase, each of these nodes starts a special version of the collect-en-route algorithm. This version collects the route information but also keeps a state at the node to indicate that the algorithm traversed it. A node that receives an additional collection message, stops the collecting process and sends the result directly to the origin using only the FF infrastructure.

The message complexity of the second phase is the sum of the path lengths from each member of the subgroup to the originator, which is bounded by Dm. The time complexity is $O(DC + D^2 P)$.

VI. Global Functions

In many management and control applications there is a need to compute a general function from variables that reside in different network elements. For example, the average load of a network, is done by retrieving local load information from the network elements and computing the average. Finding the closest DNS server, may be done by retrieving the distance of all DNS servers and choosing the one with minimal distance.

These functions are a special case of a global sensitive function [4] which we term succinct functions. These functions can be computed with a single path on the input in any order, requiring only constant memory.

The computation of succinct functions can be done using an algorithm similar to report-on-tree. Notice that when the information is collected along the tree, it is sufficient to send constant sized messages, that indicate the value of the function on the appropriate subset of nodes.

Theorem VI.1: Every global succinct function can be computed on a subset of m elements with time complexity $O(mP + DC)$ and a message complexity of $O(E)$.

Proof: Omitted. □

VII. Discussion and Further Work

This paper presents a new model that captures the hop-by-hop datagram forwarding mechanism deployed in todays IP networks, as well as the ability to perform complex computations in network elements as proposed in the active networks paradigm. The model is then used to present and analyze distributed algorithms for basic problems that arise in the control and management of IP networks.

The main point raised against enhancing routers with programmable abilities, is the alleged inefficiency and unreliability it must introduce to the network. We show that although the local delay increases due to a slower mechanism to evaluate packets, the overall performance of many important tasks significantly improves.

The suggested model can also be used to evaluate the contribution of added features to a router, and allows the formal comparison of different proposed architectures. For example, we plan to use this framework to formally analyze the power that can be gained by using the TTL mechanism, and forward and copy capability (similar to the selective copy feature of [4]). It can be easily used to analyze algorithms in IP networks, IP networks enhanced with programmable management capabilities [14], or to compare different execution environments within the active networks paradigm.

References

[1] Commentaries on "active networking and end-to-end arguments". *IEEE Network*, 12(3):66–71, May/June 1998. Edited by Thomas M. Chen and Alden W. Jackson.

[2] Y. Chae, S. Merugu, E. W. Zegura, and S. Bhattacharjee. Exposing the network: Support for topology sensitive applications. In *IEEE OpenArch 2000*, Tel-Aviv, Israel, Mar. 2000.

[3] I. Cidon and I. Gopal. PARIS: an approach to integrated high-speed private networks. *International Journal of Digital and Analog Cabled Systems*, 1(2):77 – 86, April-June 1988.

[4] I. Cidon, I. Gopal, and S. Kutten. New models and algorithms for future networks. *IEEE Transactions on Information Theory*, 41(3):769 – 780, May 1995.

[5] A. W. Group. Architectural framework for active networks. URL `http://www.cc.gatech.edu/projects/canes/arch/arch-0-9.ps`, August 31 1998. Version 0.9.

[6] C. Hedrick. Routing Information Protocol, June 1988. Internet RFC 1058.

[7] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A programming language for active networks. In *International Conference on Functional Programming (ICFP) '98*, pages 86–93. ACM, Sept. 1998.

[8] M. Hicks, J. T. Moore, D. S. Alexander, C. A. Gunter, and S. Nettles. PLANet: An active internetwork. In *IEEE INFOCOM'99*, pages 1124–1133, New York, NY, USA, Mar. 1999.

[9] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.

[10] G. S. Malkin and M. E. Steenstrup. Distance-vector routing. In M. E. Steenstrup, editor, *Routing in Communications Networks*, pages 83 – 98. Prentice Hall, 1995.

[11] S. Merugu, S. Bhattacharjee, E. W. Zegura, and K. Calvert. Bowman: A node OS for active networks. In *IEEE INFOCOM 2000*, Tel-Aviv, Israel, Mar. 2000.

[12] E. Nygren, S. Garland, and M. F. Kaashoek. PAN: A high-performance active network node supporting multiple mobile code systems. In *OPENARCH'99*, pages 78 –89, New York, NY, USA, Mar. 1999.

[13] K. Psounis. Active networks: Applications, security, safety, and architectures. *IEEE Communications Surveys*, 2(1), First Quarter 1999. `http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html`.

[14] D. Raz and Y. Shavitt. An active network approach for efficient network management. In *IWAN'99*, pages 220 – 231, July 1999.

[15] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell, and C. Partridge. Smart packets for active networks. In *OPENARCH'99*, pages 90–97, New York, NY, USA, Mar. 1999.

[16] A. Segall. Distributed network protocols. *IEEE Transaction on Information Theory*, IT-29(1):23 – 35, Jan. 1983.

[17] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan. 1997.

Appendix

I. Formal Description

In this section, we assume that a single bit is used by the filter in the FF to divert messages to the EE. We denote messages that should be diverted to the EE although the current node is not the message destination with an asterisk.

A. *collect-en-route*

The algorithm uses a single message that contains the source node id, the destination node id, and a list of ids it traverses. The source starts the algorithm by sending the message $MSG^*(s, d, \{s\})$ towards the destination, and outputs the list it receives from the destination.

collect-en-route

for $MSG^*(s, d, list)$
    if $i == d$
        send $Report(list|i)$ to s
    else
        send $MSG^*(s, d, list|i)$ to d

Fig. 7. collect-en-route for an intermediate node i

B. *report-en-route*

The algorithm uses two message types: a forward going message, MSG^*, that contains the source node id, the destination node id, and a hop counter; and a backward going $Report$. The source starts the algorithm by sending the message $MSG^*(s, d, 0)$, each node increases the hop counter by one, forward the message towards the destination, and sends a $Report$ towards the destination with its id and the hop counter value. The source uses the hop count to order the list of nodes it outputs.

report-en-route

for $MSG^*(s, d, c)$
    send $Report(ia, c + 1)$ to s
    if $i \neq d$
        send $MSG^*(s, d, c + 1)$ to d

Fig. 8. report-en-route for an intermediate node i

C. *naive*

The source sends report request messages, RR, that contain the source node id, and the destination node id. An intermediate node replies to the RR messages with a $Report$ message that contains the next hop router. $nexthop(d)$ is the function that queries the forwarding oracle for the next hop router to the destination d.

$list \leftarrow \{s\}$
$i \leftarrow nexthop(d)$
while $i \neq d$
    $list \leftarrow list|i$
    send $RR(s, d)$ to i
    receive $Report(j)$ (from i)
    $i \leftarrow j$
end while
report($list|i$)

Fig. 9. report-en-route for the source node s

for $RR(s, d)$
    $i \leftarrow nexthop(d)$
    send $Report(i)$ to s

Fig. 10. report-en-route for an intermediate node.

D. *collect-rec*

main(i, d, l)
    if $l = 0$
        return$(\{val_i\})$
    send Reach$^*_{port}(i, d, \lfloor l/2 \rfloor, \lceil l/2 \rceil)$ to d
    $L_1 \leftarrow$ main$(i, d, \lfloor l/2 \rfloor)$
    $L_2 \leftarrow$ receive$_{port}()$
    return$(L_1 | L_2)$

Fig. 11. Function main for some node i.[2]

For Reach$(s, d, count, l)$
    if $count > 0$
        send Reach$^*(s, d, count - 1, l)$
    else
        $L \leftarrow$ main(i, d, l)
        send Report(L) to s

Fig. 12. Reaction of collect-rec for receipt of message Reach()

collect(d)
    $l \leftarrow getlength(s, d)$
    $L \leftarrow$ main(s, d, l)
    return(L)

Fig. 13. Algorithm collect-rec for a source node, s.

The implementation of $getlength(s, d)$, which finds the hop length of the route between s and d, is similar to collect-en-route. Only the source sends a counter initialized to zero as the third parameter instead of an empty list. Intermediate nodes increase the counter by one instead of concatenating the next hop (see lines 3 and 5 in figure 7).

II. Proofs

A. *Proof of Theorem III.2*

Consider a line graph of n nodes, numbered $0, 1, \ldots, n - 1$. Since only local information is maintained, all nodes except, maybe one, must be queried. We claim that a node of distance $l < \frac{n}{2}$ from the source cannot be queried before time $(l - 1)(P + C)$. The proof follows by induction.

For the base case, $l = 2$, the source can send a process to node 1, where the process can query the id of node 2. Without querying node 1, the source cannot reveal the ids of node 1's neighbors. This query takes in the worst case $P + C$ time units.

Assume the hypothesis holds for $l - 1$, i.e., node $l - 1$ cannot be queried before time $(l - 2)(P + C)$. To query node l, node $l - 1$ has to be queried, and then an additional $C + P$ time units may pass, in the worst case, before the query of node l terminates.

For nodes at distance $l > \frac{n}{2}$, the source can ask the destination to start a similar process from its end. This (possibly wasteful) process can reveal the other half of the route. □

B. *Proof of Theorem V.1*

It was shown [16, Theorm PI-1-c)] that for the traditional asynchronous model the forward part of the PIF is the fastest algorithm to reach every network node. The following lemma states that this result also holds in our model.

Lemma B.1: The PIF algorithm messages reach every node before any other algorithm message initiated by the same node at the same time.

Proof: Since the message size in PIF is constant the delay bound for passing through the EE is constant as well. In addition, the lack of topology information does not allow us to use the cutthrough option, and forces any topology discovery algorithm to pass through the EE at every node it passes. Thus, the delay incurred on a packet that is sent on a link (u, v) is always the sum of the delay on the link and the delay through the EE. We can thus use the analysis for the general asynchronous model of Segall [16, Theorm PI-1-c)]. □

Proof: By Lemma B.1, the first report-direct message arrives at a node no later than any other message can be delivered to this node by any other algorithm. This first message triggers a direct message to the initiator using only the FF path. Thus, no other algorithm can generate a reply which would arrive at the initiator before the reply in report-direct. Therefor, the initiator would terminate report-direct before it can terminate any other algorithm. □

[2]Subindexing with $port$ is to indicate a possible implementation where the port number is used to deliver incoming messages to the correct recursive instantiation.