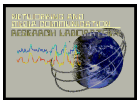


Technical Report 2003-01-02
Internetworking and Media Communications Research Laboratories
Department of Computer Science, Kent State University
http://medianet.kent.edu/technicalreports.html

COMPUTING MINIMAL SPANNING TREE WITH THE ACTIVE PROGRAMMABLE HARNESS NETWORK GROUP COMMUNICATION WARE

Javed I. Khan and Asrar U. Haque
Internetworking and Media Communications Research Laboratories
Department of Math & Computer Science, Kent State University
233 MSB, Kent, OH 44242
December 2002

Abstract The **active harness** we have developed is a scalable and versatile means for deep probing of network local states. Harness makes a clear separation between the "communication" from the "information" part of the probing process. The composition of the "information" component is handled by means of network embedded harness plug-ins. Harness supports a variety of group communication and distributed data synthesis patterns among a large set of nodes. It has been show to be capable of solving variety of messaging optimized efficient distributed algorithms used in advanced routing including shortest path, optimum clientele multicast stepping etc. In this report we illustrate how it can also solve another important problem is advanced routing—the minimum spanning tree problem. This report does not contain any performance simulation.

1.Introduction

1.1. Scalability

Scalability, in a large network, is often severely limited in point-to-point mode of communication. For example, the requesting node is required to send individual SNMP messages to all intermediate nodes for measurement of path statistics causing redundant flow of information inside the network. This increases the overhead and hence severely reduces the transparency of the measurement process. With only a point-to-point communication means, dissemination/aggregation of information creates excessive traffic on the network severely limiting the scalability. It appears that the inability to extract any intelligence from the intermediate nodes by SNMP causes the limitation. Hence, since there is no means for in network composition, all compositions must be done at the end-points, only after polling all state information there.

1.2. Versatility

On the contrary, though the specific probing kits provide greater scalability but can hardly be reused for other measurements. Nevertheless, the trend suggests that versatility of information is becoming equally important. Specifically, what measurement is useful depends on the optimization objective. For example, in a video server scenario, whether the jitter or the hard delay is more important is dependent on the specific video repair algorithm. In a different scenario, a server before sending data may want to poll information about the speed of only the last link to the home user's computers. In some other circumstances the min/max of the path downstream delays and jitters from various junction nodes can help in strategically placing jitter-absorbing

buffers in a multimedia streaming virtual private multicast network. Emerging tele-interaction applications (such as tele-surgery, remote instrument control) will require handle on the delay incurred at the video frame level, which is exactly not the same as the packet delay. The trend suggests that as more advanced and complex net-centric applications are being envisioned more versatile network state information would have to be exchanged.

1.3. Harness Approach

Can scalability and versatility both be retained simultaneously? Apparently, there may not be any efficient answer in an end-to-end paradigm. In the general case, a network can choke with polynomial messaging at the end-points. However, the recent advent of **Active Network** technology seems to offer an innovative way out from this dichotomy [13,14,8]. Active network allows programmable modules to be embedded inside network junctions. In this research we are exploring an experimental dynamic mechanism for state information polling and propagation inside network with similar embedded information synthesizers, which seems to be both scalable and versatile. The approach first makes a clear separation between the "communication" from the "information" of the state exchange and propagation process. Communication is handled by the component called "harness". Harness propagates all information via coordinated messaging. On the other hand the "information" component of the process is controlled by a set of soft programmable plug-ins. These plug-ins decide the content of the messages propagated by the harness. In a recent work [16] we have demonstrated a tree-harness system which can work on a network with tree communication topology.

In this report we present a powerful generalization of the harness that now can operate over a general graph network. The system is capable of solving various graph problems such as shortest path, max-flow etc. based on customizable criteria (such as bandwidth, delay, jitter, power usage etc.). The report explains the operation of the harness via an important network algorithm-- finding the minimal spanning tree (MST). Compared to many other problems, solution to MST is not obvious in Harness paradigm. In this report, we therefore show a harness algorithm MST that creates a scalable and customizable solution by exploiting its power of concurrent network computation and node level aggregation.

1.4. Applications of MST in Networking

Interestingly MST is finding many applications in networking. In various custom forms its use has been applied quite prominently in recent networking research. Some routing algorithms [34,35,36,37] in mobile wireless networks have used shortest-path routing where the number of hops is the path length. However, more recently researchers are suggesting that the optimum routing in wireless and mobile networking with minimum energy constraint, is a MST problem rather than shortest path [27,28,29,30,31,32,33]. Chang and Tassiullas [38] have shown that MST can be used (in inverse form) to distribute wireless traffic among various paths so that batteries of the nodes drain-out in a uniform way. Optimum mirroring based on network load reduction in web caching is also an MST problem.

MST also applies in data aggregation and distribution. Several researchers in sensor networking area have recently investigated means for scalable data aggregation [39].

The researchers used various approximations of MST. In [40] aggregation has been done based on Center at Nearest Source (CNS), Shortest Paths Tree (SPT), or Greedy Incremental Tree (GIT). PEGASIS [41] creates a chain path between network of sensors to gather and fuse data as data passes over the chain. Then fused data is sent to base station by one of the randomly chosen sensors located in the chain. Concast [52] has been proposed to merge collected feedback in multicast application. Wolf and Choi [53] proposed an aggregation algorithm similar to concast with the provision of detecting packet loss and avoidance of indefinite waiting.

MST is also central to DDBMA (Dynamic Delay Bounded Multicasting Algorithm), which is concerned about delay in multicast tree. Zhou and Hac [43] proposed a heuristic algorithm for constructing minimum-cost multicast trees with delay constraints. Other heuristics have been developed by [44,45,46]. In congestion control topology information has been used in multicast tree [47]. In Topology Aware Grouping (*TAG*) [48] the shortest path information is used to build efficient overlay networks among multicast group members.

2.Harness Architecture

The harness is in charge for initiating, propagating and responding to a series of well-coordinated messages between the nodes in a network. The harness once installed in network nodes, can act in three roles-- *session initiator*, state *synthesizer*, and *terminals*. The initiator acts as the communication agent in the network layer for the application that actually requires the information. The synthesizer propagates the state requests and processes the returning states from the terminals.

The harness controls the communication pattern and thus deals with the efficiency of messaging. Harness system accepts a set of plug-ins, which determines the content of these messages, and how they are propagated and aggregated at the junction points.

2.1. Messaging

The harness system has been designed to operate with a novel request-reply-update messaging scheme. It has three types of messages *request*, *reply* and *update*. A request message may contain fields indicating what data is needed, information dictating how far down the network the probing session should propagate either by specifying probing depth or by explicitly listing terminal nodes, and any information needed by the receiver to compute required data, e.g. to compute jitter a receiver needs to know the time stamp of sending successive data. In addition to this, for a graph structured network we need to uniquely identify a session by the combination of initiators id and some unique session id. Since in a graph structure a synthesizer might receive request for multiple sessions from adjacent nodes, the synthesizer can use the initiators id and session id to distinguish multiple sessions. The *session initiator* decides how often a request is generated in case of gathering property of a topology which changes over time. The request messages are sent to the *terminals* if they are immediately connected, or to synthesizers for further downstream propagation. A synthesizer upon receiving a request, propagates the query by generating a new request message to the down-stream nodes. However, at the same time it might also generate an immediate reply for the requestor. The replies from synthesizers may contain current local state and/or past remote states. The reply might also be used to acknowledge receipt of a request

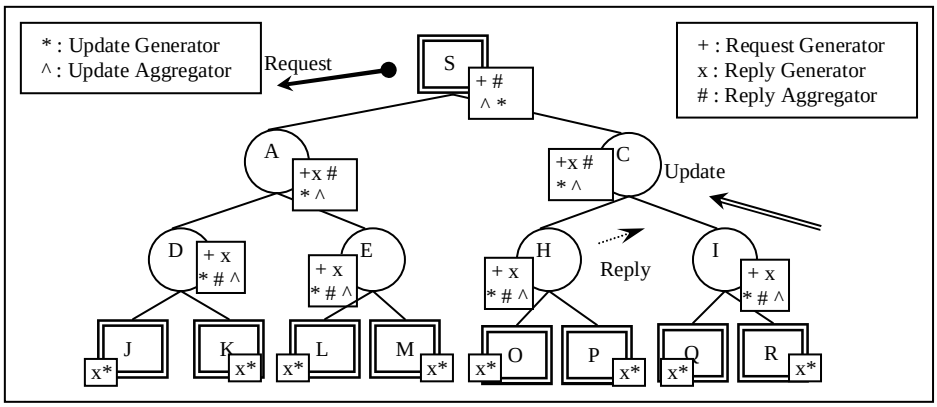


Fig. 1 Components of Active Harness

indicating that the receiver will generate request further down-stream. The terminal nodes send replies to their respective requestors. The terminal reply contains locally retrieved current states. The terminals or *update initiators* initiates return trip of information by generating update messages. In the return trip of information, the synthesizer nodes aggregate the information and at each stage generate update messages for their requestors. Once a node receives all or specific number of update messages from its immediate down-stream nodes or on timeout, it updates the network local state variables and generates a new update message. The update message contains a synthesized summary of information calculated from all its immediate downstream nodes.

This three-part request-reply-update communication model, if needed, allows the information to be collected without working in lockstep. Even if a downstream node is delayed or silent, it does not hold the entire system; the estimation process can proceed for remaining nodes. The update phase is further equipped with optional and configurable timers to avoid update lockup. In essence, the request-reply phase allows collection of local immediate states. The reply mechanism allows immediate probing into current local states and past synthesized remote states, while the update message retrieves latest remote states.

2.2. State Composition

Harness system accepts a set of six-plug-ins which are called *request generator*, *reply generator*, *update generator*, *request aggregator*, *reply aggregator*, and *update aggregator*. These modules together determine the content of these messages, and how they are aggregated at the junction points. They work via a virtual *slate*. A copy of which is maintained in each of the nodes. The slate works as the local abstract data structures. The slate is programmable and is defined at the session initiation phase. The *request generator* specifies the request message describing the fields it wants from the slate of its down-stream node. At individual nodes the model supports MIB-II and thus acts as a superset of SNMP. The terminal nodes can read/copy MIB variables (or their processed combination) existing in the local slate into variables marked for reply. The harness then invokes the reply messages with the designated slate variables. *Reply aggregator* (or *update aggregators*) in a similar fashion is invoked each time a reply (or update) is received by the harness. They perform domain specific processing of the

reply message fields and similarly update their own slate variables. The *update generator* is invoked when a special trigger variable becomes true. The trigger variable

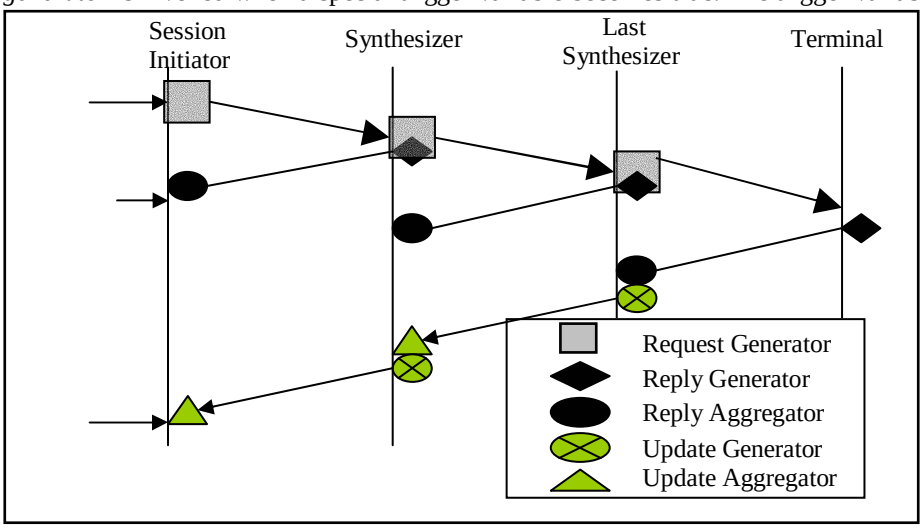


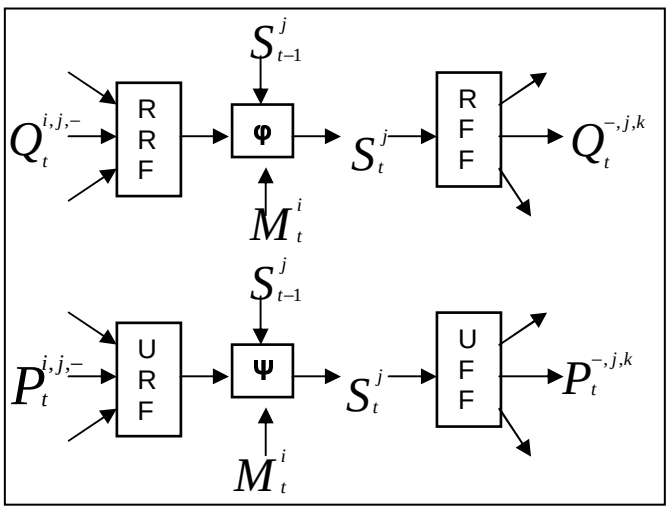
Fig. 2 Event Diagram for Harness

is a set of conditions such as all, any, or a specified number of down-stream nodes have requested/updated/replied, or a timer fires. The update generator sends the slate variables synthesized by the update aggregators to the upstream node. Fig. 1 describes the architecture of the proposed harness system. It shows the typical locations of the plug-in modules and the direction of the messages. In fig. 2, the event diagram of the harness is shown.

At the heart of the composition ability is the transfer functions of the intermediate synthesizers. The request and update phase can be represented by equations:

$$S_t^j = \Phi(\vec{E}(Q_t^{i,j,-}), M_t^j, S_{t-1}^j), \text{ and } Q_t^{-,j,k} = \vec{F}(S_t^t) \quad \ldots.(1)$$

$$S_t^j = \Psi(\overleftarrow{F}(P_t^{-,j,k}), M_t^j, S_{t-1}^j), \text{ and } P_t^{i,j,-} = \overleftarrow{E}(S_t^t) \quad \ldots.(2)$$



Here, as shown in fig 3, S_t^j is the local *slate state* at event time t at node j, $\vec{E}$ is the *request receiving* filter (RRF), M is the *local network state* (such as MIB variable), $\vec{F}$ is the *request forwarding* filter (RFF). $Q_t^{i,j,-}$ is the arrived request from parent i, to node j and $Q_t^{-,j,k}$ is the propagated requests to children k.

$\overleftarrow{E}$ is the *update forwarding* filter (UFF), $\overleftarrow{F}$ is the *update receiving* filter (URF). $P_t^{-,j,k}$ is the arrived update from child k,

and $P_t^{i,j,-}$ is the propagated update to parent i. While the filters determined the information propagation rules, composition functions $\Phi()$ and $\Psi()$ together determine the message content. While, in principle each of these components for each of the individual harness sites can be programmed differently, however the associated management will be intractable. In this harness we divide the network nodes into subsets based on their role in the topology. Nodes in the topological subsets then inherit uniform programmed behavior. Thus, we need only six distinct programmed modules (plug-ins) to be supplied by the harness programmer.

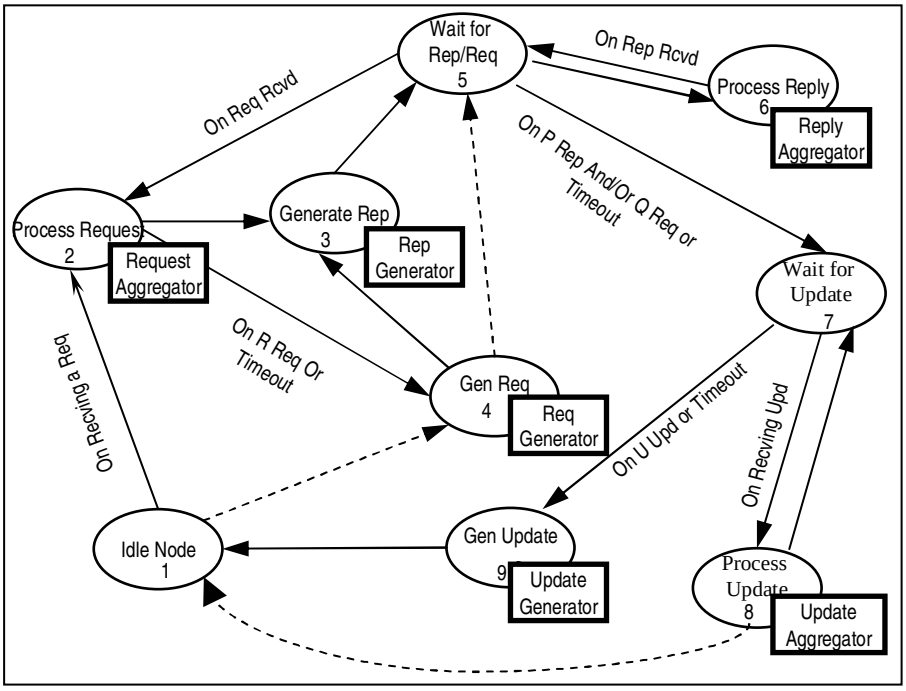
3.Harness Execution Model

The harness operates through 9 states. Fig 4 shows the state transition diagram. The oval shaped boxes describe activities and the square boxes indicate plug-in modules used for those activities. The dotted path is taken only by a session initiator. A *session initiator* has states 1-4-5-6-5-7-8-1 if reply and update is expected. An *update initiator* since waits for U=0 updates, has states 1-2-3-5-3-7-9-1. Synthesizers (if both reply and update is expected) have states 1-2-4-3-5-2-5-6-5-7-8-7-9-1. As fig 4 suggests, other combinations are possible depending on whether reply and/or update is expected or not.

mponents

4.Example Probing: Minimal Spanning Tree

Problem Definition: G(V, E) is a connected, undirected graph where V is the set of nodes and E is the set of edges and for each edge (u,v)$\in$ E, ω(u,v) is cost of edge (u,v). We need to find an acyclic subset, T $\subseteq$ E, that connects all the vertices and whose total weight ω (T) = $\sum_{(u,v)\in T} \omega(u,v)$ is minimum [54].



Fig, 4 State Transition Diagram of The Active Harness

We will map the three phases of the harness to find the minimal spanning tree of a graph topology. The first phase is request generation phase. During the request generation phase, reply phase takes place concurrently.

- A *session initiator* initiates a session of finding minimal spanning tree (MST) of a network with graph topology. How deep the probing should take place, should be determined by the *session initiator*. This can be done by identifying one or more terminal nodes (which can be thought as the leaf node of the topology) during the initiation of session. Another way of determining how deep probing should take place is the depth (or level) from the *session initiator*. However, this might lead to redundant flow of request messages across network. The *session initiator* sends request messages to its adjacent synthesizers.
- A request message sent by *session initiator* includes its id (i.e. *session initiator*s id), id of the terminal node, i.e. the *update initiator*, and a unique session number. Each session can be distinguished by a session number and id of the *session initiator*.
- When a request message is received by a synthesizer from the *session initiator* it becomes child of the *session initiator*. This synthesizer then propagates request message to its adjacent synthesizers. The new request message consists of id of the *update initiator* sent by *session initiator*, *session initiator*'s id, session number and its own id. This means that the new request message can be constructed from the

request message sent by the *session initiator* by changing sender and receivers' id and appending it own id to keep track of the path of the request propagation.

- Once a request message is received from a synthesizer, say *x*, then the sender, *x*, becomes the parent of the receiving synthesizer, say *y*, provided the receiver of the request message , i.e. *y*, has not sent a request earlier to *x*. Then the receiving synthesizer, *y*, propagates the request to its adjacent synthesizers, say *z*, from which it did not receive a request yet and z is not in the path of request message. On the other hand, if the receiver, *y*, before receiving the request message from *x*, has sent a request message to *x*, then we need to resolve the ambiguity of both being one another's parent and child. This can be resolved by following the convention that the synthesizer whose id is higher becomes the parent of the other synthesizer. This will insure that every adjacent synthesizer is either parent or a child of any other adjacent synthesizer.
- The reply message can be used to resolve/acknowledge parent-child relationship. There can be provisions so that a reply message contains previously computed MST. This can be used in situations where the synthesizers are mobile or inactive incase of energy sensitive nodes.
- As the request propagates down the links, the request messages will give the path through which request messages propagated from the *session initiator* to a synthesizer. During the request generation phase a synthesizer does not send a request message to a synthesizer which is in the path of request propagation. This can be used to avoid deadlock during update phase.
- The initiator designates a terminal which we will call *update initiator*. An *update initiator* does not send any request to any adjacent synthesizers and hence does not have any child. It initiates the update phase. However, there can be incidents where a node receives request messages from its adjacent node and becomes a child of all its adjacent synthesizer. Such a node, by definition, becomes *update initiator* even though the *session initiator* has not specified that node as an *update initiator*. We can prove that the algorithm of finding minimal spanning tree works correctly even if we have more than one *update initiator*.
- An *update initiator* initiates the update phase upon receiving request messages from its adjacent synthesizers. Update message generated by the *update initiator* includes the cost of the link between it and its parent. Essentially this is the minimal spanning tree of the sub-graph which consists of *update initiator* and its respective parent.
- During the update phase all the parent synthesizers expect an update message from each of their children. A synthesizer does not generate an update message until it receives update messages from all its children. On receiving update messages from its children a synthesizer finds the updated MST of the sub-graph which consists of that synthesizer and all its children and their children down to the *update initiator*. This can be found by merging the MST sent by its children in the respective update messages and then removing any cycle that is created by the merge operation. A cycle can be removed by eliminating the link that has maximum cost among the links in the concerned cyclic path. Once the synthesizer finds the MST of the graph below it, it chooses one of its parent synthesizers, say *m*, whose cost is the minimum among all its parent synthesizers. An update

message is sent to that synthesizer, *m*, adding the cost of that synthesizer with the newly computed MST.

- Other synthesizers which are not chosen are notified via the update message with a null path.
- Gradually, update messages reach the *session initiator* and, it computes the MST of the graph.

5.Proof of Correctness of Algorithm

Theorem: *Given, G(V, E) is a connected, undirected graph where V is the set of nodes and E is the set of edges and for each edge (u,v) ∈ E, ω(u,v) is cost of edge (u,v). Above Harness-MST algorithm gives a minimal spanning tree i.e. an acyclic subset, T ⊆ E, that connects all the vertices and whose total weight ω (T) is minimum.*

Lemma 1*: *No edge is left out without considering it.

Proof: Since through each edge a request is propagated and since a node when sending an update compares the cost of edge from which it receives a request, so all edges are considered.

Lemma 2: No less costly edge is left out (i.e. holds minimal property.)

Proof (by contradiction): Given G =(V, E), V= {u, v, z, y, x} and E={uv, vz, xy, yz, ux} and T={*uv, vz, xy, yz*} is the minimal spanning tree of graph G. We need to show that there is an edge uy with ω*(uv)* <ω*(uy)* or ω*(xy)* <ω (*ux*).

Now, there can be two scenarios, either *u* is the child of y or *y* is child of *u*. If *u* is child of *y* then while *u* generated an update message as per algorithm *u* chooses the link which has minimum cost hence weight of *uv* is less than weight of *uy*. Similarly, we can show if *y* is child of u weight of *yz* is less than *uy*. In both cases it's a contradiction. So there cannot be an edge *uy* whose weight is less than *uv* or *yz*.

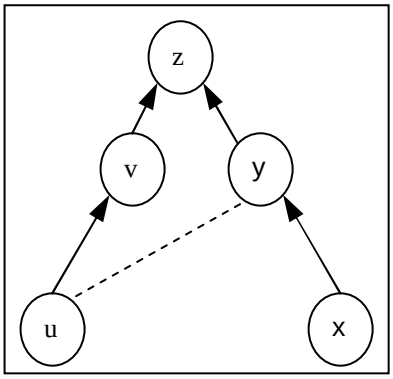


Fig. 5

Lemma 3: It is a spanning tree.

Proof: Since at every node during update generation, existence of any cycle is checked and eliminated if any, so at the end we get a tree.

Directly from lemma 1, 2, and 3, the theorem is proved.

6.Example:

In the following figures we show snapshots of finding minimal spanning tree. The numbers next to a link is the cost associated with that link, e.g. cost of node connecting nodes *a* and *b* is 4. Fig 6(a) shows the topology with cost of different links. Node *a* is the *session initiator* and *a* designates *g* as the *update initiator* (terminal node). Fig 6(b) shows the parent-child relationship between different adjacent nodes after all the

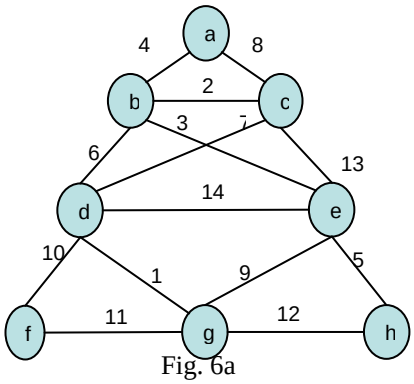


Fig. 6a

request messages have propagated down to the *update initiator*. The direction from one node to another shows the propagation of request message and parent child relationship, e.g. an edge from $a \rightarrow b$. indicates propagation of request from *a* to *b* and that *b* is a child of *a*.

The *update initiator*, *g*, on receiving requests from all its neighbors, initiates the update phase sending update message to its parents (requestors) *f*, *d*, *e*, and *h*. The update message sent to *f* includes path *gf* and cost 11. Similarly, update messages to *h* include path *gh* and cost 12. Node *f* upon receiving update message it was expecting from its child, i.e. from *g*, generates update message augmenting *fd* to *gf* and sends it to *d*. Node *d* on receiving updates from f and g, merges the paths, i.e. the respective MST, received from *f* and *g* which gives {*gf, fd, gd*} and checks if there is a cycle. If there is a cycle, it eliminates the edge with maximum weight. In this case since there is a cycle (fig 6c), it eliminates the edge *gf* whose cost is higher (i.e 11) than other two edges in the cycle. Now, remaining edges, i.e. {*fd, gd*} is the MST of the subgraph whose root is *d*. Node *d* after eliminating cycles, finds the edge *db* (6) from the edges connecting *d* to its parent which has minimum weight, and adds that to rest of the MST. This gives {*fd, gd, db*} and sends that to *b*. The other parent of *d* which is c is notified by sending another update message with null path indicating c that *dc* was not chosen. This is shown if fig 6(c). Similarly, e sends update message to *b* with the path {*gw, he, eb*}. When node *b* receives update messages from its children *d* and *e*, it detects a cycle {*gd, db, eb, ge*} and eliminates *ge* the edge with maximum weight in the cycle giving the path {*fd, gd, db, he, eb*}. After receiving update message from *c* which has {*cb*} and merging it with that which has been received from *d*, the MST of the subgraph whose root is *b* becomes {*fd, gd, db, he, eb, cb*}. Finally, *a* computes the overall MST of the topology once it receives updates from *b* and *c* which is {*fd, gd, db, he, eb, cb, ba*}.

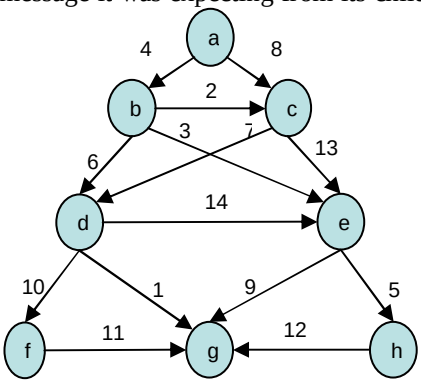


Fig 6b

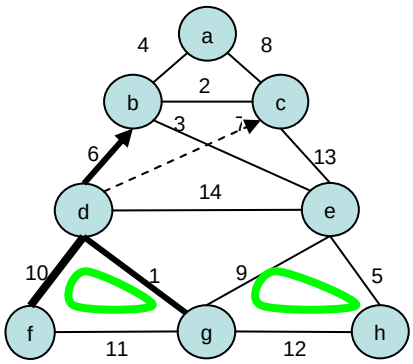


Fig 6c

7.Messages and Modules

var MsgType : type HarnessVar, var SessionNo: type SessionID, var InitiatorID: type NodeID, var UpdateInitiatorID: type NodeID, var RequestPath : type NodePtr

Fig. 7(a) Request Message

var MsgType : type HarnessVar, var SessionNo: type unsigned int, var InitiatorID: type NodeID, var Ack: type Bool

Fig. 7(b) Reply Message

var MsgType : type HarnessVar, var SessionNo: type SessionID, var InitiatorID: type NodeID, var Path: type (var PathHead:type NodePtr + var TailNodePtr: type NodePtr, var MaxLink: type NodePtr, var MaxLinkCost:unsigned int)

Fig. 7(c) Update Message

var LinkCost: array unsigned int, var PathMST: type tree

Fig. 7(d) Slate

var RequestRecvd: type array bool, var RequestSent: type array bool, var UpdateRecvd: type array bool, var UpdateSent: type array bool, var ReplyReceived : type aray bool, var ReplySent: type aray Bool, var ParentNodePtr: type NodePtr, var ChildNodePtr: type NodePtr, var RequestRecvdFromNode: type NodePtr, var RequestSentToNode: type NodePtr, var UpdateRecvdFromNode: type NodePtr, var UpdateSentToNode: type NodePtr

Fig. 7(e) Harness Variables For a Node

Request Generator Module:
For each adjacent node j
if (j not in requestMsg.RequestPath OR j has not sent request)
Set SessionNumber, InitiatorID, UpdateInitiatorID
Path ← append (requestMsg.RequestPath , parentNode)
>> **Send request message to j** <<

Fig. 7(f) Request Generator Module

Request Aggregator Module:
For each adjacent node j
if (r=1 request received)
signal (activate Request Generator Module)

Fig. 7(g) Request Aggregator Module

```
Reply Generator Module :
    if (request not sent to either to  j OR to node in RequestPath )
        set replyMsg.Ack ← True
    otherwise
        set replyMsg.Ack ← False
    >> send reply message to j <<
```

Fig. 7(h) Reply Generator Module

```
Reply Aggregator Module:
    if (reply msg from j.ack = False )
    set ParentNodePtr (next) ← j
     otherwise
    set ChildNodePtr (next) ← j
```

Fig. 7(i) Reply Aggregator Module:

```
Update Generator Module:
        For each parent node  j of current node i
                set MsgType, SessionNo, InitiatorID
        if (j.Linkcost = min)
                update message.Path ← cons (j, Slate.PathMST)
        else
                update message.Path ← NULL
              >> send update message to j <<
```

Fig. 7(j) Update Generator Module:

```
Update Aggregator Module:
    For update message from j
        Slate.PathMST← merge (Slate.PathMST, update message.Path)
            if (cycle in Slate.PathMST) then
                Slate.PathMST← remove (Slate.PathMST, maxLink
(Slate.PathMST)
```

Fig. 7(k) Update Aggregator Module:

8.Analytical Complexity

MST is one of the oldest known graph problems having an illustrious history. Textbook algorithms run in $O(E \lg N)$ time, where N and E denote, respectively, the number of vertices and edges in the graph. A distributed algorithm was presented in [49] that construct the minimum-weight spanning tree in a connected undirected graph with distinct edge weights. The total number of messages required for a graph of N nodes and E edges is at most 5NlgN+2E. As we will show that the proposed harness

algorithm will use much less number of messages, at most 3E. On average total number of bits on all the messages will depend on number of bits used to identify nodes.
For a graph G = (E, V) maximum number of request and reply messages is same which is 2|E|. The number of update message is |E| during update phase from *update initiator* towards the *session initiator.*
During request generation, a request generator need has complexity O(*db*) in worst case where *d* is the depth of that node and *b* is the average branching factor. It takes O(b^2), in worst case, in order to detect a cycle for an update generator. So for an update phase if depth of *update initiator* is O(*d*) then it takes O (db^2).
In classical solutions by Kruskals, Prims etc for the MST problem assume that the graph topology in known and hence the complexity *O*(*m* lg *n)* is estimated only on the basis of the computation of MST. Such centralized solution however ignores the cost of data collection, which can be substantial O(*nd*) in a network environment with n nodes. Our approach drastically reduces the amount of information propagated. This is important in large scale Internet environment where communication is much significant cost factor than the small computations involved.

Table-1 Link Traffic Impact

Type	Byte	Max Msg	Msg/link	Bytes/link
Request	r=O(*d*)	|E|	1	r=O(*d*)
Reply	p= O(1)	|E|	1	p= O(1)
Update	u=O(*d*)	|E|	1	u= O(*d*)

Table-2 Node Processing Impact

Node	Request		Reply		Update	
	Generator	Aggregator	Generator	Aggregator	Generator	Aggregator
Initiator	O(*b*)	O(*b*)	0	O(*b*)	0	O(b^2)
Synthesizer	O(*bd*)	O(*b*)	O(1)	O(*b*)	O(*b*)	O(b^2)
Terminal	0	O(*b*)	O(1)	0	O(*b*)	0

Above we have provided estimates of the impact on the links and on the nodes due to the harness operation respectively for one execution/propagation wave. Here, as mentioned before, *d* and *b* stand for a maximum probing depth and average branching factor of the context network respectively. The sizes of the Request, Reply, and Update messages are of r = O(*d*), p = O(1), and u = O(*d*) bytes respectively and computational impact in a synthesizer due to *request generator*, *request aggregator*, *reply generator*, *reply aggregator*, *update generator* and *update aggregator* are r_g = O(*db*), r_a = O(*b*), p_g= O(1), p_a = O(*b*), u_g = O(*b*), and u_a = O(b^2) respectively. Table-1 shows the network wide traffic, message per link and the byte density. The potential scalability of the system is indicated by the message density per link which is 1. Table-2 shows the computational impact on the network nodes due to the plug-ins. It can be noted here that, a particular state-probing session may be launched with a subset of capabilities (such as no reply, but update). The design objective is to provide the least impact communication for the given application scenario. With the timeout feature we can

approximate MST of a graph in O(*d*) time which can be good enough for many practical purposes.

9.Conclusions

The report presents the Harness system via an example of its use in solving MST. The key to the system's **scalability** and **versatility** are the embedded aggregators. Since local state dependent aggregation is performed inside a network, it reduces communication and thus enhances the system's scalability. Aggregators also provide the ability to compute network relative deep composite statistics, over the elementary MIB-II variables, thus enhancing the versatility of its ability to collect network states.

The network implementation is non-trivial nevertheless can be realized at user space as *deamons*. Embedded implementation can cut down some overhead and will be critical for sub-second range probing cycles. Implementation on some form of active platform [8,13,14] can further facilitate matters such as remote deployment, and seamless secured execution of the plug-ins. The harness plug-ins require very limited form of programmability compared to general active net paradigm. Also, the read-write suggestions are through local slate variables only. These characteristics assuage many of the security concerns. The proposed harness is perhaps one of those cases where provisioning even very low-grade programmability can be highly rewarding. The harness increases state visibility of network. In effect it facilitates high pay off smart optimizations for numerous applications, which are not easily realizable today due to the black box nature of current network. Interestingly, such a network layer utility is not only crucial for building a new generation of network aware applications but it is also vital for many of the current problems internet is grappling with. Interestingly many of which are arguably artifacts of the opacity of current network design. Currently, we are exploring its active network based simulation. The work is being supported by the DARPA active network Research Grant F30602-99-1-0515.

10.References

1. Carter, Robert L., Mark E. Crovella. *Measuring Bottleneck Link Speed in Packet-Switched Networks*. Performance Evaluation 27 & 28 (1996), 297-318.
2. Comer D. E., Internetworking with TCP/IP, Principles, Protocols, and Architectures, 4th Ed, Pretice Hall, New Jersey, USA, ISBN- 0-13-018380-6, 2000
3. Dong, Yingfeng, Yiwei Thomas Hou, Zhi-Li Zhang, Tomohiko Taniguchi. *A server-based non-intrusive measurement Infrastructure for Enterprise Networks*. Performance Evaluation (36)1, 1999, pg 233-247.
4. Downey Allen B., *Using Pathchar to Estimate Internet Link Characteristics*. http://ee.lbl.gov/nrg-talks.html, April 1997.
5. Paul Francis , Sugih Jamin , Cheng Jin, Yixin Jin , Danny Raz , Yuval Shavitt , Lixia Zhang, *IDMaps: a global internet host distance estimation service*. IEEE/ACM Transactions on Networking (TON) October 2001 Volume 9 Issue 5
6. Jacobson, V., *Traceroute*, [URL: ftp://ftp.ee.ibl.gov/traceroute.tar.Z,] 1989.
7. Jacobson, V, *Pathchar- a tool to infer characteristics of Internet paths*, [URL: http://ee.ibl.gov/nrg-talks.html], April 1997.
8. Javed I. Khan, S. S. Yang, *Medianet Active Switch Architecture*, Technical Report: 2000-01-02, Kent State University, [available at URL http://medianet.kent. edu/technicalreports.html, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet]

9. Matt Mathis, Jamshid Mahdavi. Diagnosing Internet Congestion with a Transport Layer Performance Tool. Proceedings INET, 1996, Montreal Canada.
10. Paxson, V., Jamshid Mahdavi, Andrew Adams and Matt Mathis. *An Architecture for Large-Scale Internet Measurement*. IEEE Communication Magazine, August 1998, 48-54.
11. Jeffrey Case, Rob Frye, Jon Saperia. *SNMPv3 Survival Guide*, October 1999 Published by John Wiley & Sons
12. Schulzrinne, H., S. Casner, R. Frederick, and V. Jacobson. *RTP: A Trasport Protocol for Real Time Applications*, RFC 1889, 1996.
13. Tennenhouse, D. L., J. Smith, D. Sincoskie, D. Wetherall & G. Minden.. *A Survey of Active Network Research*. IEEE Communications Magazine, Vol. 35, No. 1, Jan 97, pp 80-86
14. Wetherall, Guttag, Tennenhouse. ANTS: *A Tool kit for Building and Dynamically Deploying Network Protocols*. IEEE OPENARCH'98, San Francisco, April 1998. Available at: http://www.tns.lcs.mit.edu/publications/openarch98.html
15. Javed I. Khan & S. S. Yang. Resource *Adaptive Nomadic Transcoding on Active Network*, Applied Informatics. AI 2001, February 19-22, 2001, Insbruck, Austria, [available at URL http://medianet.kent.edu/, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet]
16. Javed I. Khan & Asrar U. Haque. *An Active Programmable Harness For Measurement of Composite Network States*. IEEE International Conference on Networking, ICN' 2001, Colmer, France, June 2001, pp628-638
17. D. Makofske and K. Almeroth, *MHealth: A Real-Time Multicast Tree Visualization and Monitoring Tool. Network and Operating System Support for Digital Audio and Video (NOSSDAV '99)* , Basking Ridge New Jersey, USA, June 1999
18. A. Swan and D. Bacher, *rtpmon 1.0a7*. University of California at Berkeley, January 1997. Available from ftp://mmftp.cs.berkeley.edu/pub/rtpmon/
19. K. Almeroth, *Multicast Group Membership Collection Tool (mlisten)*. Georgia Institute of Technology, September 1996. Available from http://www.cc.gatech.edu/computing/Telecomm/mbone/
20. J. Robinson and J. Stewart, *MultiMON 2.0 – Multicast Network Monitor*, August 1998. Available from http: // www.merci.crc.ca/ mbone/ MultiMON/.
21. D. Thaler, *Mstat*. Merit Network, Inc. and University of Michigan. http://www.merit.edu/net-research/mbone/mstat.html
22. D. Thaler and A. Adams, *Mrtree*. Merit Network, Inc. and University of Michigan. http://www.merit.edu/ net-research/ mbone/mrtree man.html
23. D. Thaler, *Mview*. Merit Network, Inc. and University of Michigan. http://nic.merit.edu/mbone/mviewdoc/Welcome.html.
24. K. Lai and M. Baker. Nettimer: *A Tool for Measuring Bottleneck Link Bandwidth*. In Proc. of USENIX Symposium on Internet Technologies and Systems, March 2001.
25. Kamil Sara , Kevin C. *Scalable techniques for discovering multicast tree topology*. 11th International workshop on Network and Operating Systems support for digital audio and video January 2001
26. Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble, *King: Estimating Latency between Arbitrary Internet End Hosts*. *To appear in the Proceedings of SIGCOM IMW 2002*, Marseille, France, November 2002.
27. Dennis J. Baker and Anthony Ephremides, *The architectural organization of a mobile radio network via a distributed algorithm*, IEEE Transactions on Communications, vol. COM-29, no. 11, pp. 56–73, Jan. 1981.
28. Anthony Ephremides, Jeffrey E. Wieselthier, and Dennis J. Baker, *A design concept for reliable mobile radio networks with frequency hopping signaling*, *Proceedings of the IEEE*, vol. 75, no. 1, pp. 56–73, Jan. 1987.
29. Volkan Rodoplu and Teresa H. Meng, *Minimum energy mobile wireless networks*, in Proceedings of the 1998 IEEE International Conference on Communications, ICC'98, Atlanta, GA, June 1998, vol. 3, pp. 1633–1639.
30. Teresa H. Meng and Volkan Rodoplu, *Distributed network protocols for wireless communication*. In Proceedings of the 1998 IEEE International Symposium on Circuits and Systems, ISCAS'98, Monterey, CA, June 1998, vol. 4, pp. 600–603.
31. S. Singh, M.Woo, and C.S. Raghavendra, *Power-aware routing in mobile ad hoc networks*, In Proceedings of Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking, Dallas, TX, Oct. 1998, pp. 181–190
32. Timothy Shepard, *Decentralized channel management in scalable multihop spread spectrum packet*

radio networks, Tech. Rep. MIT/LCS/TR-670, Massachusetts Institute of Technology Laboratory for Computer Science, July 1995

33. M. Ettus, *System capacity, latency, and power consumption in multihop routed SS-CDMA wireless networks*. In Proceedings of IEEE Radio and Wireless Conference *(RAWCON) 98*, Colorado Springs, CO, Aug. 1998, pp. 55–58
34. C. Perkins and P. Bhagwat, *Highly dynamic destination-sequenced distance vector routing (DSDV) for mobile computers*. In *ACM SIGCOMM*, Oct. 1994
35. S. Murthy and J.J. Garcia-Luna-Aceves, *An efficient routing protocol for wireless networks*. ACM Mobile Networks and Applications Journal, Special Issue on Routing in Mobile Communication Networks, 1996.
36. D. Johnson and D. Maltz, *Dynamic source routing in ad hoc wireless networks*. Mobile Computing, 1996.
37. Vincent D. Park and M. Scott Corson, *A highly distributed routing algorithm for mobile wireless networks*. In Proc. IEEE INFOCOM'97, Kobe, Japan, 1997.
38. Jae-Hwan Chang and Leandros Tassiulas, *Energy Conserving Routing in Wireless Ad-hoc Networks*. INFOCOM '2000, March 2000.
39. C. Intanangonwiwat, D. Estrin, R. Govindan, *Directed Diffusion: A scalable and robust communication paradigm for sensor networks*. In Proceedings of 6th ACM/IEEE MobiCOM Conference, 2000.
40. B. Krishnamachari, D. Estrin, and S. Wicker, *Modelling Data-Centric Routing in Wireless Sensor Networks*. In Proc. of IEEE Infocom, 2002
41. S. Lindsey and C. S. Raghavendra, *PEGASIS: Power Efficient GAthering in Sensor Information Systems*. In Proc. of IEEE Aerospace Conference, 2002
42. Moustafa Youssef, Mohamed Younis, and Khaled Arisha, *A Constrained Shortest-Path Energy-Aware Routing Algorithm for Wireless Sensor Networks*. Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), Orlando, Florida, March 2002
43. Anna Hać Kelei Zhou, *A New Heuristic Algorithm for Finding Minimum-cost Multicast Trees with Bounded Path Delay*. International Journal of Network Management Volume 9, Issue 4 July–August 1999
44. Kou L, Markowsky G, Berman L. *A fast algorithm for Steiner trees*. *Acta Information* 1981; 15:141–145
45. Kompella VP, Pasquale JC, Polyzos GC. *Multicast routing for multimedia communication*. IEEE/ACM Transactions on Networking 1993; 1(3):286–29
46. N. M. Malouch, Z. Liu, D. Rubenstein, and S. Sahu, *A Graph Theoretic Approach to Bounding Delay in Proxy-Assisted, End-System Multicast.* In 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'02), May 2002
47. S. Jagannathan and K. Almeroth, *Using Tree Topology for Multicast Congestion Control*. In *Proc. of International Conference on Parallel Processing*, September 2001
48. Minseok Kwon, Sonia Fahmy. *Distribution Overlays: Topology-aware overlay networks for group communication*, International Workshop on Network and Operating System Support for Digital Audio and Video, Proceeding of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video, 2002
49. R.G. Gallager, P.A. Humblet, and P.M. Spira, *A distributed algorithm for minimum weight spanning trees*, Tech. Rep. LIDS-P-906-A, Lab. Inform. Decision Syst., Massachusetts Inst. of Technol., Cambridge, MA,Oct. 1979
50. Kamil Sara , Kevin C. Almeroth, *Scalable techniques for discovering multicast tree topology*, 11th International workshop on on Network and Operating Systems support for digital audio and video January 2001
51. Mohamed Younis, Moustafa Youssef, and Khaled Arisha, *Energy-Aware Routing in Cluster-Based Sensor Networks*, Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'02), Fort Worth, Texas, October 2002
52. Kenneth L. Calvert, James Gri.oen, Billy Mullins, Amit Sehgal, and Su Wen, Concast, *Design and implementation of an active network service*. *IEEE Journal on Selected Areas of Communications*, vol. 19, no. 3, pp. 404–409, Mar. 2001
53. Tilman Wolf and Sumi Yunsun Choi, Aggregated Hierarchical Multicast for Active Networks. *Aggregated hierarchical multicast for active networks, Proceedings of IEEE MILCOM*, October 2001
54. Thomas Cormen, Charles Leisterson, Ronald Rivest, and Cliff Stein. *Introduction to Algorithms, McGraw Hill Publishing Company and MIT Press, 2001.*