DANS: Decentralized, Autonomous, and Network-wide Service Delivery and Multimedia Workflow Processing *

Gisik Kwon
Computer Science and Engineering
Arizona State University
Tempe, Arizona, USA
gkwon@asu.edu

K.Selçuk Candan
Computer Science and Engineering
Arizona State University
Tempe, Arizona, USA
candan@asu.edu

ABSTRACT

Fundamental challenges in designing environments with media-rich ambient services involves not only the development of appropriate sensing technologies, but as importantly, the implementation of a distributed media processing system which can process, integrate, and leverage the sensed data in real time to provide the various services. In recent years, a great deal of progress has been made in media service workflow processing systems. In most existing solutions, however, the workflow nodes, which operate on the data, are selected out of a centrally assigned candidate pool. These candidate organizations cause either extensive resource provisioning or poor-quality operator mapping between logical workflow nodes and the available physical resources nodes. Consequently, instantiating a media processing workflow to the underlying hardware before the workflow execution begins does not lends itself to adaptive and autonomous operation of the workflow, scalable to resources and demand.

In this paper, we propose a novel decentralized multimedia workflow processing system, *DANS*, in which operators defined in workflows are mapped into (distributed) physical nodes through Distributed Hash Table (DHT)-based overlay substrate in a purely decentralized and adaptive manner. The redundancy in the system, in terms of availability of multiple nodes able to perform the same task, enables the system to scale with demand. Furthermore, physical workflow nodes (operator instances) are able to locate and select the next filter or fusion operator instance autonomously, while ensuring the correct execution of the workflow.

Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Distributed Applications; C.4 [Performance of Systems]: Reliability, availability, and serviceability; J.5 [Arts and Humanities]: Performing arts

General Terms

Algorithms, Performance, Reliability, Experimentation

Keywords

Stream workflow processing, Media processing workflows, Peer-to-peer computing

*This work has been supported by NSF grant # IIS-0308268, "Quality-adaptive Media-flow Architectures to Support Sensor Data Management."

MM'06, October 23–27, 2006, Santa Barbara, California, USA.

1. INTRODUCTION

Networks are becoming ubiquitous and networking capabilities are being integrated into an increasing diversity of equipments, ranging from home consumer electronics to smart dust type sensing networks. Thereby, visions of the near future refer to an Ambient Intelligence [1], where sensors embedded in an environment will capture and notify relevant events and ubiquitous services will adapt to the current situation.

ARIA, ARchitecture for Interactive Arts [2, 3], an ambient intelligence system we are developing at ASU, uses multimedia processing workflows to describe how data sensed in an intelligent performance stage will be filtered, fused, and processed and responses will be actuated to support interactive dance performances. Such environments providing ambient, situation-aware services, require cognitive capabilities transparently embedded in the surroundings to (a) continuously sense the environment, target status, and the context, (b) filter and fuse multitude of real-time media data, and (c) react by appropriately adapting the environment. The fundamental challenge in designing media-rich ambient intelligence involves not only the development of appropriate sensing technologies, but more importantly the implementation of a distributed media processing system which can process, integrate, and leverage the sensed data in real time to provide the various services. In particular, the ambient service software and hardware have to adapt the environment (through actuators which deliver the services) as well as itself (appropriately allocating resources, processing elements, and quality of service).

We note that environments with ambient intelligence, such as smart rooms, smart offices, and smart class-rooms, have different structures and components; yet, they all need to process various media collected through environmental sensors and react by actuating appropriate responses. Designing such an open reactive system is challenging as run-time situations are partially known or unknown in the design phase and multiple, potentially conflicting, criteria have to be taken into account during the runtime. How tasks are structured, who performs them, what their relative order

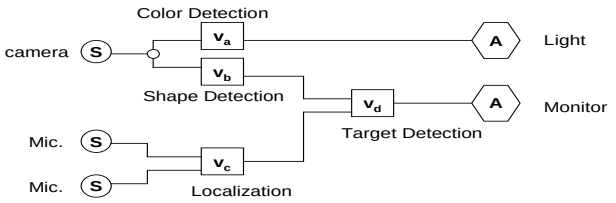
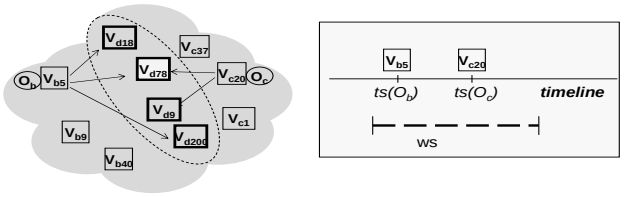


Figure 1: An ARIA media processing workflow[2, 3]



(a) (b)

Figure 2: (a) A collection of distributed operator instances executing a portion of the workflow in Figure 1, and (b) two objects generated within windowing time. The source operator instances, v_{b5} and v_{c20}, must independently choose an operator instance enforcing the windowing constraint among all available v_d operator instances.

is, how they are synchronized, and how information flows between processing units to support the tasks are usually described in terms of *workflows*. Therefore, designing of media-rich ambient services require (a) the description and semantic integration of service delivery workflows consisting of adaptable sensing, processing, communicating, and actuating components, (b) modular integration of various media processing and actuating components into service delivery workflows and adaptation of these workflows for providing end-to-end service delivery guarantees, and (c) run-time workflow evolution and adaptation to changing resources and network characteristics.

The inherent context-based adaptation and distributed nature of the resources require that the workflow processes are continuously mapped and remapped to different clusters of resources without interrupting the ongoing services. Thus, in this paper, *we focus on this last challenge of enabling adaptation of the media processing workflow delivery in a media-rich sensory/reactive environments.*

1.1 Enabling Adaptation of the Media Processing Workflow Delivery

Media-rich ambient services with sensory-reactive functionalities requires various sensing components to collect and process different forms (audio, video, radio) of environmental data in real-time. In many existing systems, the underlying sensing and processing logic is hardcoded into the hardware and software. This, however, limits the portability, interoperability, and evolution of the media-rich ambient services. This is a major limitation and a roadblock against the widespread deployment of ambient services.

Therefore, we propose to address this challenge by firstly breaking the dependence to hardcode in real-time media processing. In particular, we propose to describe sensory-reactive services as adaptable workflows, composed of individual media sensing/processing/actuating units such as Figure 1. More details regarding media processing workflows are provided in Section 2.1.3. A media processing workflow combines various operators, in large sensor networks, which sense, filter, transform, and fuse media objects in real time. Sensors act as object sources. Filter and fusion operators provide analysis, aggregation, and filtering semantics. In-network data analysis, through such distributed media processing workflows, both enable the right information at the right quality with the minimal transmission cost and provide scalability and robustness through replication of available processing nodes.

Traditionally, distributed stream workflow processing systems (DSPSs) collect, process, and aggregate data across large numbers of data sources supporting applications as various as financial market monitoring and network intrusion detection. Unlike such conventional data and workflows, a media processing workflow needs to capture the continuous nature of the processing needs due to streaming characteristic of the sensory data, redundancy and imprecision in media and media processing units, and the quality of service requirements and other constraints at the various levels of the workflow.

A fundamental consideration in DSPSs is the mapping of a logical workflow plan onto the distributed physical nodes in the network. Recently, there have been a number of efforts in the area of automatic composition of network wide processes (SpiderNet [4], SAHARA [5, 6], SPY-Net [7], CANS [8], and Infopipes [9]). Unlike these works, which focus largely on network services, our focus is to address challenges associated with enabling adaptation of the arbitrary media processing workflow delivery itself. An approach to sensory data management involves a centralized warehouse as in TelegraphCQ [10]. In this scheme, all streams must be routed to the central warehouse, thus this approach does not lead to an efficient use of resources; instead, in a media processing workflow, this would cause extensive resource provisioning including centralized points of failure, intrinsic to such client-server systems. To address this shortcoming, recent works, such as SAND [22] and PIER [15], propose to leverage routing paths in a distributed hash table (DHT) to identify a set of candidate nodes for operator instance mapping[1]. We provide more details regarding the related work in streaming data management in Section 7. However, here we note that the less flexible operator-instance to hardware mapping provided by this approach does not provide true adaptivity in distributed ambient services, where the resource availability and demand can be highly variable.

Also, the approach of instantiation of workflows prior to the workflow execution, relied upon in most of these systems, does not enable adaption of the workflow delivery in runtime. Thus, a decentralized operator mapping scheme along with an adaptive and autonomous workflow routing protocol are critical. Yet these tasks are also very difficult, without a dedicated coordination, especially when there are various constraints (such as windowing constraints of the operators) are to be enforced (Figure 2). These are the challenges we are tackling in this paper.

[1]In this paper, "*operator*" stands for the logical node in a workflow, whereas "*operator instance*" stands for a hardware or software component capable of executing the logical operator semantics.

1.2 Contributions of this Paper

In this paper, we propose a novel Decentralized Autonomous and Network-wide Service workflow processing system(*DANS*). This system executes the requested workflow in a distributed manner over a potentially heterogeneous network. In particular, we begin to address the problems faced in media-rich DSPSs with exploring DHT-based peer-to-peer systems as a low-level overlay substrate in a purely distributed and adaptive manner. These DHT-based systems [23, 24, 25, 26] are known to enable computing networks in which participating nodes cooperate to share computing bandwidth and load in a purely distribute manner, while providing the high scalability and the node availability. Thus, they will help with large-scale decentralized node arrangements and with autonomously locating the next node in the workflow. This system executes the requested workflow in a distributed manner. DANS runs over a P2P substrate, responsible for the maintenance of connectivity between resource nodes, the management of node membership, and the lookup and publishing of node information.

In DANS, the network-adaptivity is achieved through sender-initiated operator mapping, as opposed to a the receiver-initiated process used in most current DSPSs. In the receiver-initiated approach [22], the workflow instantiation to physical nodes is accomplished before the workflow execution starts. This requires high maintenance costs due to need for failure detection and recovery of nodes and for migration of processed in execution. On the other hand, in the sender-initiated approach we introduce in DANS does not require pre-establishment of workflow networks. The workflow instantiation begins when source nodes start injecting objects into the network. An operator is chosen *by the sender* among qualified candidates currently active in the network, providing high adaptivity to the network changes and eliminating the overhead of maintaining instantiated workflow networks.

According to the design considerations stated above, the main contributions made in this paper includes followings:

- we present a novel decentralized DSPS architecture, *DANS*, which organizes resource pools and executes workflows in a purely decentralized manner.
- we provide an effective, yet efficient operator search mechanism which assures qualified operators to be found to process objects.
- we introduce a sender-initiated and object-based workflow instantiation, providing high adaptivity to the underlying network and alleviating the initial resources and maintenance cost.

The rest of this paper is organized as follows. Section 2 provides an overview and states the problem. Section 3 describes overall design and presents the construction and operation of DANS. Section 4 and 5 states how DANS leverages windowing constraints in the workflow to reduce communication cost. Section 6 evaluates the performance and overheads of DANS. Section 7 reviews the related work. Section 8 concludes this paper and talks about future works.

2. DANS OVERVIEW

In this section we provide the requirements deriving the development of the DANS decentralized media processing workflow system and the constraints underlying its design.

2.1 Media Processing Workflows

Various high level workflow languages (UML activity diagrams, WSFL) exist but are not applicable to the design of media-rich ambient environment design. In media processing domain, tools (such as Max/MSP and Pure Data (Pd), are used by media engineers to design media-specific workflows; however, these tools are limited in expressive power and workflows designed using these limited software tools are geared towards soft applications (such as art installations) that are lenient to centralized and best-effort deployments. In particular, they lack the capability of delivering and executing rich media processing and fusion operators.

2.1.1 Operators and Objects

In DANS, media processing workflows describe how a specific media processing task can be performed by combining various available operators, such as sensors, filters, fusion operators, communication operators, and environmental actuators: a filter takes a stream as input, processes and transforms the objects in the stream, and outputs a new stream consisting of the transformed objects; a fusion operator takes as its input multiple streams and returns as its output multiple streams; and actuators cause changes in the environment or in the workflow itself. Each data object, O, exchanged between a pair of operators contains note only its payload, but also meta data which describes the objects various properties. Depending on the task, the object payload ranges from a simple numeric value to a complex image component segmented out from frames in a video sequence. The object header is augmented by each operator. It contains timestamps to check the temporal requirements, and the source node identifier to distinguish an object from other ones when an operator executes multiple services at a time.

2.1.2 Object Queues and Windowing Constraints

An operator, v, has input and output queues: $Iq(v) = \{in\_q_i, 1 \leq i \leq k\}$ and $Oq(v) = \{out\_q_j, 1 \leq j \leq l\}$. The operator picks a set of objects from each input queue and processes them, and outputs the results into the output queue. Naturally, Each queue in the system has limit on the number of objects. A fusion operator, Φ, has a number of characteristics that determine its runtime properties, including an *input selection model.* This model answers the question "which possible candidate combination in the inputs will be processed at a given point in time?" The way the operator picks its inputs is generally governed by a time-constraint on the input objects: due to the continuous nature of the input streams fusion operators are usually implemented in a time-windowed manner: given a window size, ws, the timestamps (ts) of objects being fused should satisfy the following constraint:

$\forall in\_q_i, in\_q_j$ of a fusion operator, Φ, $o_i \in in\_q_i$ and $o_j \in in\_q_j$, should be considered for fusion if and only if, $|ts(o_i) - ts(o_j)| < ws$, where $ts(o_i)$ denotes the time stamp the object o has when it is injected into the network by a prior operator.

2.1.3 Media Processing Workflow Graph

A media processing workflow, w, is a directed graph; $G(V, E, C)$, where vertices, V, represent the operators; edges, E, represent the connections between the operators; and constraints, C, represent a set of constraints (such as the windowing constrains above) that the workflow should enforce [2, 3]. An example workflow is shown in Figure 1: in

this figure, a multimedia data processing workflow is used to drive visual effects with object streams originated from camera and microphones. In this workflow, camera sends objects to two filters, v_a and v_b, to detect the color and shape; microphones send objects to fusion operator, v_c which translate raw data into formatted ones. Shape and localized objects are directed to target detector, v_d, which finally delivers the generated picture to monitor actuator.

2.2 Decentralized Workflow Execution

The rapid growth of processing power and networked communication enables large number of individual machines to participate into a large-volume of cooperative work as servers. Technologies associating such contributive machines with candidate server pools in a distributed manner have been well investigated [23, 24, 25, 26]. These infrastructures reduce the maintenance cost of complex systems and provide of adaptivity.

2.2.1 Autonomy and Adaptivity

However in the case of distributed media processing workflow execution, there are additional challenges that have to be considered. For example, sensory/reactive systems generally execute on continuous inputs conveyed from sensors. Therefore such workflows should run uninterrupted for long times under the proviso that the execution should be independent, unobtrusive, and adaptive against changes in physical network layers and servers.

What we have to keep in mind, on the other hand, is that to enable adaptivity fore-mentioned services have to be deployed in a decentralized environment, meaning without any dedicated coordination helper. Therefore a major challenge DANS has to handle involves how the filter and fusion operators are mapped into the physical nodes in the underlying network in a distributed manner. Once the mappings between the heterogeneous abstractions are provided, large volumes of streams can flow from sensors to actuators with no provisioned servers. A related challenge to decentralized workflow delivery is that each operator in the workflow needs to make autonomous and adaptive decisions to find an appropriate destination to operate the resulted outputs. This task is especially hard for finding fusion operators, as two or more source nodes have to choose (independently) the same fusion node to operate on their output objects.

2.2.2 DANS Layers

In DANS, operator instances make autonomous decisions to locate next physical nodes in the workflow and the execution of workflow proceeds independently from the changes of underlying network. To enable both of these goals in a decentralized and adaptive manner, DANS consists of three loosely-coupled layers that provide well-defined functions linking hardware resources to workflows (Figure 3):

- Hardware Resource Layer (HRL) organizes a virtual network through which computing entities can participate freely within local or from wide-area networks.

- Operator Instance Layer (OIL) not only maintains the operator instances (i.e. software code) which can be instantiated to logical operators defined in the workflow, but also maps each operator instance to a physical node in the hardware resource layer.

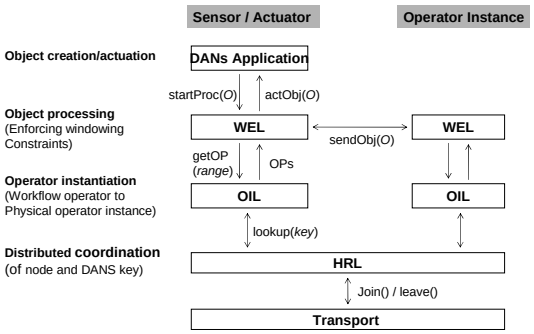


Figure 3: DANS System Structure: In DANS, workflow execution layer runs independently from lower layers, which endows each operator with autonomous and adaptive decision making to select a destination. Vertical arrows are local APIs; horizontal arrows are inter-node APIs.

- Using the operator instances provided by OIL, the Workflow Execution Layer (WEL) processes the input objects along with enforcing the windowing constraints specified in the workflow.

While these three layers collaboratively enable operations of DANS, it is important to note that the workflow execution layer runs independently from the lower layers (except for requesting for an appropriate operator instance), which endows each operator with autonomous and adaptive decision making capabilities for selecting the next node in the workflow. The architecture of DANS is discussed in more detail in Section 3.

2.2.3 Autonomous Next Operator Instance Selection and Windowing Constraints

If an object is sent to a node which cannot satisfy the underlying windowing constraints, then the object will simply be ignored (dropped from the queue). Naturally this is not desirable; i.e., objects should be directed to fusion operators where they can satisfy appropriate windowing constraints. One naive solution to this could be to send the object to all possible operator instances. Yet, this could be tremendously expensive because of the network traffic that can be caused by the resulting redundancy. Thus an efficient and effective prediction technique to select an appropriate set of candidate nodes is essential to alleviate these costs, without causing a large number of ghost fusion operations, where objects are dropped simply because the chosen fusion operators can not enforce the windowing constraints. This issue is discussed in Section 3.2.1 in more detail.

3. SYSTEM ARCHITECTURE

In this section, we present the DANS architecture. We also describe algorithms used in next-operator instance selection process during workflow execution and demonstrate the process with a simple DANS network example.

3.1 Structure of DANS

In DANS, the registered workflows are either stored in a fixed location or replicated over the network to shed reference loads. Each node obtains the workflow information, such as operation types and execution constraints, looking

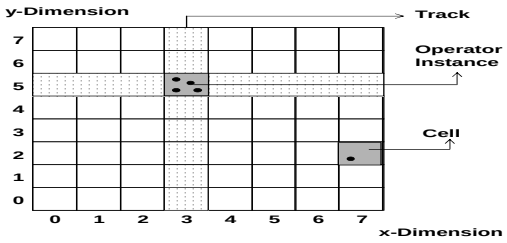


Figure 4: Operator Instance Pool represented as a grid: The operator mapping associated with an operation type takes place within the corresponding operator pool.

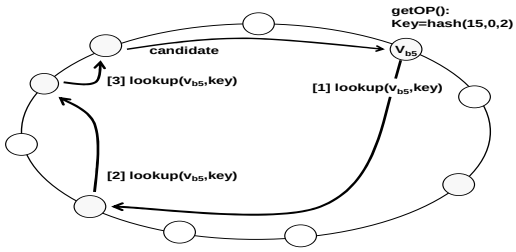


Figure 5: Operator Instance Lookup through Chord. Chord processes a candidate lookup in $O(logN)$ with the DANS coordination key.

through these registries. In this paper, we do not focus on the registration process, but on the operator instance to node mapping and the workflow execution processes.

Given a set of registered workflows, DANS acts as a system of pools of candidate operator instances, supporting the various operations defined in the media processing workflows. Naturally, each operator mapping takes place in the operator instance pool associated with the required workflow operators. In DANS, each operator instance pool is organized in the form of a hyper-cube, where the number of dimensions correspond to the number of input streams to the operator. For example, as shown in Figure 4, for a fusion operator with two input streams, the operator instance pool would be organized as a two-dimensional grid, in which each participating node is randomly mapped into a cell.

3.1.1 Hardware Nodes and Operator Instances

When a new hardware node is available to perform one or more of the listed operations to serve, simple screening procedures are taken to check whether the node is eligible for the operation. The node considers the capacity of general computing power, like network bandwidth and memory size, and specific resource requirements, such as the existence of the necessary hardware/software components. This static screening ensures that the node meets the bare minimum qualifications for the operation. Once the node chooses operations to execute, it acts as an operator instance for those operations. The chosen operation type(s) indicates the candidate operator instance pools in which the node will be considered.

3.1.2 Operator Instance to Grid Mapping

To publish and retrieve the operator directory on top of a P2P substrate, slight modifications are made to the primitive procedures in the substrate, including join and lookup. In the join process, a step to form DANS-grid is added. For efficient support of distributed operator-instance lookup, *DANS* combines DHT-based lookup and publication substrate (*Chord* [23] in this paper) with the hardware resource layer. While joining into the system, each node picks an identifier used for DHT-based lookup and publish. This identifier is a hashed value of the node's IP_address. Each node creates a key for the node directory and publishes it along with the context of DHT substrate.

As described before, for an operator with k inputs streams, the directory is organized as a grid with k-dimensions. Therefore, for each operator instance, the key is hashed using the node information: [*operation type to serve*, $coord\_value_1$, $coord\_value_2$, ..., $coord\_value_k$]. The directory insertion and retrieval in Chord can be implemented with the directory key, formed through the same hash function which produces node identifier. The directory key is stored at the successor node of the key identifier, and, in turn, the successor of the key identifier is queried for the directory retrieval. When using Chord as the DHT substrate, this lookup operation requires $O(logN)$, where N is the total number of nodes in the system. To preserve the invariant that directory is stored at the successor of their associated keys, hardware resource layer monitors the arrival and departure of nodes using the callback interface provided by Chord and moves directory in he underlying node-space appropriately. When a hardware node leaves or join the system, the successor node of a given key may change.

Figure 5 demonstrates the Chord DHT lookup process in a simple DANS network. In the example, source node v_{b5} requests a list of candidates to serve the operation type 15, located at the grid coordinate (0,2). The details of the Chord lookup process is outside of the scope of this paper.

A set of cells sharing the same dimension coordinate are referred to as a *track*. In each *next-operator instance selection round*, discussed in Section 3.2, DANS picks a range of tracks and searches the candidate nodes in those tracks.

3.1.3 Grid Size and its Impact on Service Delivery

As described above, the operator instance-to-cell mapping is randomized. Thus, each cell in the grid may contain zero or more nodes according to the join and leave rates of operator instances. Since, on the average, each cell is expected to contain a single operator instance, the number of cells in the grid should be close to N, i.e., the number of operator instances. Thus, given an operator with k input streams, the size of the each track (i.e., the number of cells along each dimensions) is $N^{\frac{k-1}{k}}$. Naturally, the operator instance pool size N can change during the operation of DANS. If the deviation of the actual number of operator instances from the expected number of operator instances is large, this can lead to undesirable situations: First, widespread empty cells in the grid can cause high failure rates and, thus, can increase the network cost for the searches required to fix the problem. Excessive number of nodes in a single cell can also render operator selection process unscalable due to large replication overhead in each cell. DANS addresses these potential challenges through on-demand restructuring as well as through a flexible next-operator selection process, as discussed in the next subsection.

3.2 Autonomous Selection of the Next Operator Instance

As discussed in Section 2.1, in multimedia processing workflows, two objects are compatible for fusion if they satisfy the *windowing constraint* associated with the fusion operator:

> Two objects o_i and o_j are fusible if they are created within ws time unit from each other; i.e., if $|ts(o_i) - ts(o_j)| < ws$.

Let $oi_{src,1}$ and $oi_{src,2}$ be two operator instances that have generated one object each (almost) simultaneously. Let us also assume that, according to the workflow objects generated by these operator instances have to be fused together. One of the main tasks of DANS is to enable $oi_{src,1}$ and $oi_{src,2}$ to *independently* choose the next operator instance without having to first find and talk with each other.

For instance, in Figure 2, objects generated by operators v_b and v_c are fused, by operators v_d. Thus, operator instance for v_{b5} should be able to pick the next fusion operator instance of type v_d without having to communicate first with all v_c operator instances. As shown in Figure 7 (a), DANS relies on the intersection-of-tracks property of the proposed grid structure to enable independent decision making for v_{b5} and v_{c20} in chosen the next operator instance. In this section, we discuss in detail how DANS enables its nodes to achieve this task autonomously; in particular we provide an explain the protocol used by the operator instances. The pseudo code in Figure 6 presents the outline of the main algorithms used in the next operator instance selection process.

3.2.1 Outline of the Next Operator Instance Selection Process

Next operator selection starts when a new object is generated by an operator instance, oi_{src} and this object needs to be sent to an instance, oi_{dst}, of the next operator in the workflow. The workflow execution layer provides the **startProc()** service interface which implements the next-operator selection service.

Naturally, the output stream of oi_{src} corresponds to the one of the input streams of oi_{dst}. Since each input stream of a fusion operator is represented by a dimension of the corresponding grid, the source operator instance need to pick one or more tracks on the receiving operator's grid along the corresponding dimension. The number of tracks used by the source is referred to as the *range* and denoted as r; in Section 3.2.2, we will discuss how operator instances select the initial range value. Thus, at each round, oi_{src}

1. randomly picks a set of tracks corresponding to the initial range r.
2. identifies the candidate operator instances along the chosen track(s) via operator instance layer's getOP(). Upon receipt of the candidate track range from the workflow execution layer through the getOP() service call, the operator instance layer
   (a) creates the directory key of each candidate cell,
   (b) requests a lookup for the key to the hardware resource layer,
   (c) stores the retrieved candidate node information to *cand_list*, and
   (d) returns the list to the workflow execution layer as the result of the getOP() call.

   If there is any cell group, with no suitable node along the chosen tracks, the source nodes start another search, only around this cell group, with a larger range (Figure 7 (b)). Cell group is a set of cells in the track range sharing the same other dimension coordinates. The process proceeds until at least one eligible operator instance is found along each cell group on the chosen tracks.
3. sends requests to all candidate operator instances in the track. The request contains the object header, with the timestamp of the objects. Essentially, each operator instance along the track has been marked with the id of the operator as well as time stamp of the object. This timestamp will expire after ws time units from the time it is set.
4. as oi_{src} receives object-request replies from the candidate fusion operator instances (i.e., the operator instance oi_{dest} has already a suitable object for fusion or

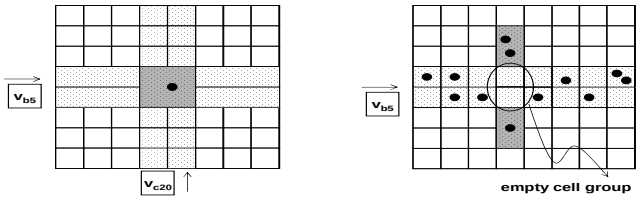
```
N = size of operator instance pool
width = size of a dimension, identical in DANS

startProc(object)
  repeat
    s_tr=rand()%width
    range=ceil(width/^k sqrt(N))
    e_tr=s_tr+range-1
    cand_list=getOP(s_tr,e_tr)//first round
    for (empty cell group) getOP(empty cell group) //second round
    for (i=0;i<|cand_list|;i++)  send object to candidate_i
    if (|reply|>0)
      winner = tie_breaking(replied candidates)
      send object to winner
      return success
    else
      drop the object
      return fail

getOP(s_tr,e_tr)
  origin = source_node_ip_address
  for (i=0;i<width;i++)
    for (j=s_tr;j<e_tr;j++)
      key = hash(op_type,i,j)
      cand = lookup(origin,key)
      push cand to cand_list
  return (cand_list)

lookup(origin,key)
  check local table for highest node, target, s.t., my_id<target<key
  if (target exists)//intermediate node
    lookup(origin,key) to node target
  else//final lookup node
    get latency between origin and target //added in DANS
    return (target, latency)
```

Figure 6: Next Operator Instance Selection Process in DANS: In these algorithms, we use k-dimensional operator instance pool.



(a) (b)

Figure 7: Track Intersection: (a) DANS relies on the intersection-of-tracks property enabling the autonomous candidate selection, and (b) DANS ensures the property by extending the range of an empty cell group.

oi_{dest} receives one such object before the timestamp expires after ws time units), it sends the object to at most one requester per cell, note that sending the object to multiple requesters per cell can cause redundant computations.

In this scheme the windowing constraints are enforced by the receiving node, using the timestamps set by the object sources. Thus, this scheme assumes that the clocks of the source and the destination operator instances are synchronized. Especially, in a distributed environment, this may be an overly constraining requirement. In Section 5, we will provide an alternative scheme, which does away with this requirement.

3.2.2 Selecting the Initial Range based on Density of Suitable Nodes

The track range, *range* used at the beginning of the first search round depends on the grid size and density, as discussed in the previous Section 3.1. Using a large initial range will provide a higher success rate, but will also cause a large network cost. Let us consider a grid of k dimensions with N operator instances. With the operator instances uniformly distributed in the grid and when the grid size is chosen in such a way that each cell will have ~ 1 operator instance, a range of 1 will cost $O(N^{\frac{k-1}{k}})$ communications along the given track and will mark $O(N^{\frac{k-1}{k}})$ nodes. Note that, when $k = 2$ (which is highly common), we have $O(\sqrt{N})$ communication cost.

If DANS frequently identifies empty cells, then it means that the grid is larger than the number of available operator instances required. Thus, the grid has to shrink. However, without waiting the grid to shrink, DANS lets each operator instance adaptively modify the initial *range* parameter. In particular, DANS implements three *range* selection approaches: the incremental, decremental, and r-steady. The incremental approach stretches the range either linearly or exponentially if there are lots of empty cells along the chosen track. The decremental approach shrinks the range in the same manner, if the cells return significantly larger number of operator instances than expected.

In r-steady the source computes the appropriate range directly based on the available statistics (history of the number of nodes retrieved from track in the past accesses) rather than continuously modifying it. As described above, given N operator instances and a grid of size $width^k$, where $width$ is the cells along each dimension, the number of expected nodes in each cell is $\frac{N}{width^k}$. Therefore, we need $(\frac{N}{width^k})^{-1} = \frac{width^k}{N}$ cells to have 1 operator instance. If every source operator instance use the value r as the range (computed based on common statistics), then r^k should be equal to $\frac{width^k}{N}$; i.e., the initial range chosen by each source operator instance should be $r = \lceil \frac{width}{N^{\frac{1}{k}}} \rceil$.

4. PRUNING CANDIDATE OPERATOR INSTANCES THROUGH WINDOWING CONSTRAINTS

The basic DANS architecture described above can be used for workflow executions with windowing constraint through timestamps and markers that are valid only for the duration of window size, ws. However, we note that it is possible to use the windowing constraints to reduce the number of candidate nodes that has to be marked by source nodes.

For example, trivially, any node which will take more than ws time units to locate and send the object can be ignored as by the time that the object reaches the corresponding operator instance, the corresponding marker will be too old and thus the object can not join with any object at this operator instance anyhow. Thus, if the distance, Δ, between oi_{src} and oi_{dest}, is greater than ws, that oi_{dest} can be pruned. To support this, the latency information is obtained from an external service. Let us assume that only λ of the N operator instances are within ws units from oi_{src}. To ensure that each cell-range along the track will have ~ 1 suitable operator instance, the initial range oi_{src} will use need to be at least $\frac{N}{\lambda}$ wide.

5. TTL-WINDOWING AND PRUNING

Although the above scheme can help eliminate the those candidates that can not satisfy the windowing constraints based on the basic DANS protocol, if λ is very small or for most available candidates Δ is very close to ws, the protocol can cause missed fusion opportunities. An alternative, time-to-live (TTL) based, windowing scheme would eliminate such missed opportunities as follows:

1. Let o_1 be generated by a source operator instance at t_1 and forwarded to a destination. Let the minimum time it takes to communicate between the source and destination be Δ_1^{min}. Then, the earliest time o_1 is received by the destination is $t_1 + \Delta_1^{min}$.

2. Let o_2 be an object which is generated by another source operator instance within the time window ws from t_1. The latest time o_2 may be generated is $t_2 = t_1 + ws$.

3. Let also the maximum time it takes to communicate between the source of o_2 and destination be Δ_2^{max}. Then, the latest time o_2 is received by the destination is $t_1 + ws + \Delta_2^{max}$.

Therefore, the time-to-live of the marker at the destination operator instance can be computed by

$$TTL = (t_1 + ws + \Delta_2^{max}) - (t_1 + \Delta_1^{min}) = ws + \Delta_2^{max} - \Delta_1^{min},$$

where Δ_2^{max} is the largest temporal separation between the destination operator instance and any possible generator for

objects that can fuse with o_1. Note that with this TTL-based marker scheme, it is possible to put an arbitrary limit on Δ_2^{max}; i.e., we can let the sources of objects, o_2, to prune their destinations based on a fixed threshold, $\Delta_2^{\top}$ and we can let the destination nodes adapt the TTLs for objects, o_1, based on this chosen threshold. Once again, if such a threshold is chosen, the source operator instances for o_2 would need adapt their initial range in such a way that each cell-range in the grid along the chosen tracks will contain ~ 1 operator instances.

6. EVALUATION

To the best of our knowledge, DANS is the first truly decentralized and operator-autonomous multimedia workflow execution system. In this section, we first validate the system in terms of correctness of its operations (Section 6.2) and then we analyze the performance gains it provides (in Section 6.3). In particular, we present results for the basic DANS protocol as well as the proposed TTL-based pruning of the operator instances.

6.1 Simulation Set-up

In order to observe the performance of DANS in an inherently heterogeneous distributed environment, we organized the underlying network infrastructure using the GT-ITM model [28] which provides a transit-stub network topology. In this simulation, we generate three different sizes of resources: 1K, 4K, and 8K resource nodes. These nodes are used for running the various operator instances. In the experiments, we used an average inter-transit domain delay of 1500 ms, inter-transit node to 500ms, transit domain to stub domain to 100ms, and intra-stub node to 20ms.

In the experiments, we considered a decentralized workflow execution scenario with ~ 16 different operator types and around 256 operator-instances for each (i.e., around one operator instance per node for the 256 resource pool size). The actual number of nodes available at any point in time varies from 1K to 8K as mentioned previously. This leads to three representative cases of node populations per cell: rare-, normal- (i.e., ~ 1), and the excessive-densities, respectively.

For the experiments, we considered fusion operators with two input streams. For these operator types, we need a 2-dimensional operator instance pool (i.e., $k = 2$). We iterated the experiment 1000 times. At each iteration of the experiment, a pair of source operator instances generate objects that need to be merged. To consider the worst case scenario, these pair of objects are always generated to be within the window size, WS. We also assumed an r-steady track range, computed based on available statistics beforehand as opposed to incremental modification of the operator-instance pool structure.

Basic DANS configuration sends requests to all available candidates and assigns a marking duration with WS. TTL-DANS sends requests to all retrieved candidates but assigns a marking duration of WS+$\Delta_2^{\top}$-Δ_1^{min}. In the rest of the paper, we refer to these configurations as *Basic-* and *TTL-DANS* configurations.

6.2 Validation of DANS

We validate DANS using an object processing ratio measure defined as, $opr = \frac{\#\ processed\ object\ pairs}{\#\ total\ object\ pairs}$. As mentioned in the simulation set-up, the source operator-instances generate only qualified objects (i.e., objects that are within the window size). Therefore, for each request injected into DANS, at least one qualified candidate operator instance should exists in the network. Therefore, DANS should be able to locate this suitable node.

In order to verify the existence of the suitable node, we also implemented an exhaustive operator instance search mechanism (referred to as EXH). This mechanism is guaranteed to find the suitable operator instance, if at least one operator instance eligible for the processing exists in the system. Using this exhaustive scheme, we can ensure that, in case DANS fails to find an operator instance, then there are no eligible operator instance exists in the network. In fact, since, for every task, a candidate is guaranteed to exist, we expect EXH to return 100% *opr*. Thus, if correct, DANS should also have 100% *opr* value.

In Figure 8, we present two experiment setups with different windows size (WS) values: 5000 and 1700. For these experiments, the inter-transit domain delay is set to be 1500. Note that this implies that for the WS=5000 case, the scope of valid candidate operator instance selection can stretch across three transit domains. In contrast, when WS is 1700, the scope for search should be limited within a single transit domain.

The first thing to notice in this figure is that, in terms of *opr*, both EXH and TTL-DANS achieve 100% object processing ratio. That is, TTL-DANS is able to locate a suitable fusion operator in each and every case.

The basic DANS scheme on the other hand fails to return qualified operator instances in many cases. This is especially true when the operator instance pool and/or the window sizes are large. This is obviously counter-intuitive. However, it is easy to explain: when the operator instances to be considered are large (either due to large number of available resources or due to the size of the acceptable neighborhood), basic DANS spends more time for the search of the suitable nodes. Since in the basic DANS, the timeclock is never stopped from the point of generation of the objects, this leads to cases where although there exist suitable operator instances, finding them cost more time then the window size itself. Therefore, the TTL-DANS which does not require clock synchronization and which can account for network delays explicitly performs much better (in terms of not missing potential operator instances) than basic DANS. In short, we would not recommend to use the basic DANS in situations where network delays are comparable to window sizes.

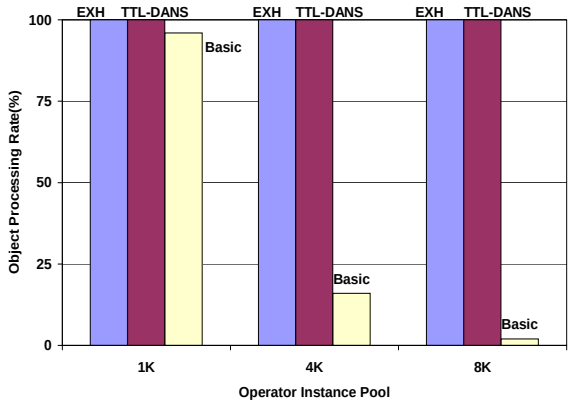
6.3 DANS Performance Gains

In this section, we show that the message overhead that DANS requires to locate a suitable operator instance is significantly small when compared to the alternative mechanism (EXH) for communicating with all available operator instances.

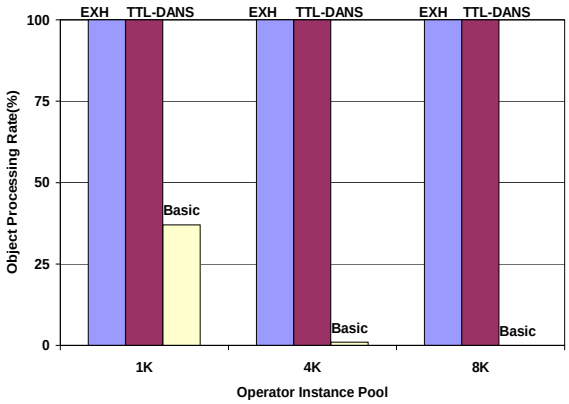
The communication cost of DANS can be estimated as

$$k \times log(N) \times r \times oi\_pool\_size^{1/k} + k \times m,$$

where k is the dimensions of the operator instance pool grid (2 in the experiments), N is the number of resource nodes, r is the range (1 or 2 in the experiments), $oi\_pool\_size = \frac{N}{num\_operator\_types}$ is the number of operator instances of a given type, $\sqrt[k]{oi\_pool\_size}$ is the number of cells to consider along a track for this operator instance, and m is the total number of nodes that are marked. The savings therefore can



(a) WS=5000



(b) WS=1700

Figure 8: DANS Correctness of Operation

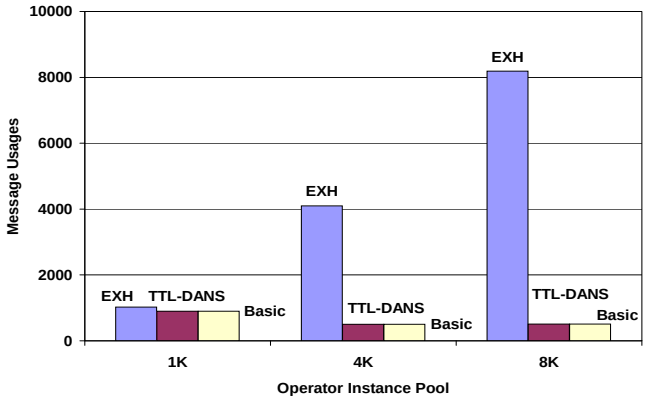


Figure 9: Network Gains in DANS

be achieved if the above value is less than N; i.e., the total number of resource nodes that an exhaustive mechanism that does not rely on DANS would have needed to check.

As shown in Figure 9, in these experiments, the exhaustive searches overhead is significantly higher than that of DANS. This is true for most cases, as $log(N)$ and $\sqrt[k]{oi\_pool\_size}$ are likely to be significantly lower than N in general.

Note that although the number of transmission required by DANS itself is seemingly high, most of these transmission can be made in parallel. The search delay itself is dominated by the DHT lookup process; i.e., $log(N)$. Therefore, especially in cases where $log(N)$ is large and a small delay is required, we recommend operator instances to avoid DANS lookups for each and every new object; instead, once a suitable operator instance is found for a pair of sources through the first lookup, this pair should maintain contact with each other through the same suitable fusion operator instance for the future pairs of objects. These sources should look for a new destination if there is a significant change in the generation rate of the objects.

7. RELATED WORK

In this section, we give an overview of work related to path optimization in stream-based overlay networks. Distributed stream processing systems, such as Medusa [11] and Borealis [12], are faced with the path optimization problem when placing stream operators. Data paths in Medusa are controlled by contracts for load management [13] that take node properties into account but are not network-aware. In the work on network-aware query processing [14] for Borealis, operators are either placed at the consumer side, at the producer side, or in-network on the DHT routing path, depending on the network bandwidth usage for a query. Applications can also specify delay constraints on the placement path in the DHT. By performing in-network placement in latency space, our approach has more freedom in choosing a good placement node without having the restriction of following DHT routing paths. In addition, relaxation placement supports dynamic, global path optimization across circuits and considers node and link stress.

Distributed databases like PIER [15] partitions data across multiple nodes. The location of operators (and corresponding relational tables), a distributed database built on top of a DHT, is determined through hashing. Such a random distribution has good load-balancing properties but uses the underlying network resources inefficiently. Mariposa [16] is a distributed database system that follows a market-based approach with economic techniques for a distributed implementation of the query optimization problem. However, global network costs, such as network utilization, are not addressed. More recent efforts on rate-based query optimization [17] and intelligent partitioning of streams [18] are directly applicable to our work when making local placement decisions for services.

Content distribution networks build an overlay network for efficiently disseminating data to many parties, which requires a network-aware selection for the location of multicast nodes. Narada [19] is an application-level multicast (ALM) system, which builds a multicast tree out of an overlay mesh. The authors introduced the issue of network efficiency and defined metrics to quantify resource usage, which are similar to ours. The overlay mesh is optimized dynamically but cannot be optimized at the granularity of a single circuit as in our approach because the mesh is shared among applications. Overcast [20] is an ALM scheme, which maximizes total bandwidth for content distribution but does not deal with global network utilization. Publish/subscribe systems, such as Scribe [21], are built on top of a DHT and rely on hashing for routing path selection.

8. CONCLUSION

In this paper, we presented the DANS architecture for truly decentralized and adaptive execution of media processing workflows in sensory/reactive environments providing ambient services such as ARIA [2, 3]. In particular, we show that the *operator instance placement* as well as the *next operator instance selection* tasks can be achieved in a purely distributed manner, without needing a centralized decision maker or a lookup directory. We also presented mechanisms to enforce windowing constraints without requiring clock synchronization and showed how to leverage these constraints to reduce communication costs during *next operator instance selection* and save resources. This is in stark contrast with existing centralized and pre-mapped approaches which can not adjust to the changes in network and resource requirements. Future research will involve investigation of mechanisms for other quality of service guarantees in decentralized media processing workflow delivery.

9. REFERENCES

[1] M.S.Raisinghani, A.Benoit, J.Ding, M.Gomez, K.Gupta, V.Gusila, D.Power, and O.Schmedding. *Ambient Intelligence: Changing Forms of Human-Computer Interaction and their Social Implications.* Journal of Digital Information, Volume 5 Issue 4 Article No. 271, 2004.

[2] Lina Peng, K. Selcuk Candan, Kyung D. Ryu, Karamvir S.Chatha, and Hari Sundaram. *ARIA: An Adaptive and Programmable Media-flow Architecture for Interactive Arts.* ACM Multimedia Conference, Interactive Art Program, New York, October 2004.

[3] K. Selçuk Candan, Gisik Kwon, Lina Peng, and Maria Luisa Sapino. *Modelling Adaptive Media Processing Workflows.* IEEE International Conference on Multimedia and Expo (ICME), July, 2006.

[4] X.Gu and K.Nahrstedt. *Distributed Multimedia Service Composition with Statistical QoS Assurances.* IEEE Transactions on Multimedia, 2005.

[5] B.Raman and R.H.Katz. *An Architecture for Highly Available Wide-Area Service Composition.* Computer Communication, 26(15):1727–1740, September 2003.

[6] B.Raman and R.H.Katz. *Load Balancing and Stability Issues in Algorithms for Service Composition.* IEEE INFOCOM, 2003.

[7] D.Xu and K.Nahrstedt. *Finding Services Paths in a Media Service Proxy Network.* MMCN, 2002.

[8] X.Fu, W.Shi, A.Akkerman, and V.Karamcheti. *CANS: Composable, Adaptive Network Services Infrastructure.* Proc. of USITS, March, 2001.

[9] A.Black, J.Huang, R.Koster, J.Walpole, and C.Pu. *Infopipes: An Abstraction for Multimedia Streaming.* Multimedia Systems 8: 406-419, 2002.

[10] S.Chandrasekaran, O.Cooper, A.Deshpande, and M.Franklin. *TelegraphCQ: Continuous Dataflow Processing for an Uncertain World.* The first Biennial Conf. on Innovative Data SystemsResearch (CIDR), Asilomar, CA, 2003.

[11] M.Cherniack, H.Balakrishnan, M.Balazinska, D.Carney, U.Cetintemel, Y.Xing, and S.Zdonik. *Scalable Distributed Stream Processing.* The first Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, 2003.

[12] D.Abadi, Y.Ahmad, and H.Balakrishnan. *The Design of the Borealis Stream Processing Engine.* Technical Report CS-04-08, Brown University, 2004.

[13] M.Balazinska, H.Balakrishnan, and M.Stonebraker. *Contract-Based Load Management in Federated Distributed Systems.* NSDI, San Francisco, CA, 2004.

[14] Y.Ahmad and U.C.Cetintemel. *Network-Aware Query Processing for Stream-based Applications.* The 30th Int. Conf. on Very Large Data Bases (VLDB), Toronto, Canada, 2004.

[15] R.Huebsch, J.M.Hellerstein, and N.Lanham. *Querying the Internet with PIER.* The 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, 2003.

[16] M.Stonebraker, P.M.Aoki, and W.Litwin. *Mariposa: A Wide-Area Distributed Database System.* VLDB Journal, vol. 5(1), 1996.

[17] S.D.Viglas and J.F.Naughton. *Rate-based Query Optimization for Streaming Information Sources.* Int. Conference on Management of Data (SIGMOD), 2002.

[18] M.A.Shah, J.M.Hellerstein, S.Chandrasekaran, and M.J.Franklin. *Flux: An Adaptive Partitioning Operator for Continuous Query Systems.* The 19th International Conference on Data Engineering (ICDE), Bangalore, India, 2003.

[19] Y.H.Chu, S.G.Rao, and H.Zhang. *A Case for End System Multicast.* ACM SIGMETRICS, Santa Clara, CA, 2000.

[20] J.Jannotti, D.K.Gifford, and K.L.Johnson. *Overcast: Reliable Multicasting with an Overlay Network.* The 4th Symposium on Operating Systems Design and Implementation (OSDI), San Diego, CA, 2000.

[21] M.Castro and M.Jones. *An Evaluation of Scalable Application-Level Multicast using Peer-to-peer Overlay Networks.* INFOCOM, San Francisco, CA, 2003.

[22] P.Pietzuch, J.Shneidman, M.Roussopoulos, M.Seltzer, and M.Welsh. *Path Optimization in Stream-Based Overlay Networks.* Harvard University Technical Report TR-26-04, 2004.

[23] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan. *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications.* Proc. ACM SIGCOMM 2001, pp.149-160, 2001.

[24] B.Y.Zhao, J.D.Kubiatowicz, and A.D.Joseph. *Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing.* UC Berkeley,UCB//CSD-01-1141, 2000.

[25] A.Rowstron and P.Druschel. *Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems.* Proc. 18th IFIP/ACM International Conference onDistributed Systems Platforms (Middleware 2001), 2001.

[26] S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker. *A Scalable Content-Addressable Network.* Proc. ACM SIGCOMM, 2001.

[27] D.Eastlake. *RFC 3174 - US Secure Hash Algorithm1 (SHA1).* 2001.

[28] K.Calvert and E.Zegura. *GT Internetwork Topology Models (GT-ITM).* College of Computing,Georgia Institute of Technology, 1996.