Utility Accrual Channel Establishment in Multihop Networks

Karthik Channakeshava, *Student Member*, *IEEE*, Binoy Ravindran, *Senior Member*, *IEEE*, and E. Douglas Jensen, *Member*, *IEEE*

Abstract—We consider Real-Time CORBA 1.2 (Dynamic Scheduling) *distributable threads* operating in multihop networks. When distributable threads are subject to time/utility function-time constraints, and timeliness optimality criteria such as maximizing accrued system-wide utility is desired, utility accrual real-time channels must be established. Such channels transport messages that are generated as distributable threads transcend nodes, in a way that maximizes system-wide, message-level utility. We present 1) a localized utility accrual channel establishment algorithm called Localized Decision for Utility accrual Channel Establishment (or LocDUCE) and 2) a distributed utility accrual channel establishment algorithm called Global Decision for Utility accrual Channel Establishment (or GloDUCE). Since the channel establishment problem is $\mathcal{NP}$-complete, LocDUCE and GloDUCE heuristically compute channels, with LocDUCE making decisions based on local information pertaining to the node and GloDUCE making global decisions. We simulate the performance of the algorithms and compare them with the Open Shortest Path First (OSPF) routing algorithm and the optimal algorithm. We also implement these algorithms in a prototype testbed and experimentally compare their performance with OSPF. Our simulation and experimental measurements reveal that GloDUCE and LocDUCE accrue significantly higher utility than OSPF and also perform close to the optimal for some cases. Furthermore, GloDUCE outperforms LocDUCE under high downstream traffic.

Index Terms—Real-time systems, multihop networks, real-time channels, time/utility functions.

1 Introduction

The recent introduction of OMG's Real-Time CORBA 1.2 standard [1] (abbreviated here as RTC2) and Sun's upcoming Distributed Real-Time Specification for Java [2] specify *distributable threads* as the programming and scheduling abstraction for system-wide, end-to-end scheduling in real-time distributed systems.[1] A distributable thread is a single thread of execution with a globally unique identifier that transparently extends and retracts through local and remote objects. Thus, a distributable thread is an end-to-end control flow abstraction, with a logically distinct locus of control flow movement within/among objects and nodes. In the rest of the paper, we refer to distributable threads as *threads* except as necessary for clarity.

A thread carries its execution context as it transits node boundaries, including its scheduling parameters (e.g., time constraints, execution time), identity, and security credentials. Hence, threads require that Real-Time CORBA's *Client Propagated* model be used. The propagated thread context is used by node schedulers for resolving all node-local resource contention among threads, such as that for a node's physical (e.g., CPU, I/O) and logical (e.g., locks) resources, and for scheduling threads to optimize system-wide timeliness. Thus, threads constitute the abstraction for concurrency and scheduling.

RTC2 describes several approaches for thread scheduling, called *Distributed Scheduling: Cases 1, 2, 3, and 4*. We consider the Case 2 approach, according to which, node schedulers use the propagated thread scheduling parameters and independently schedule thread segments on respective nodes to optimize the system-wide timeliness optimality criterion. Thus, scheduling decisions made by a node scheduler are independent of other node schedulers. Though this results in approximate, global, system-wide timeliness, RTC2 supports the approach due to its simplicity and capability for coherent end-to-end scheduling.

1.1 TUFs and UA Scheduling

In this paper, we focus on dynamic real-time systems at any level(s) of an enterprise—e.g., in the defense domain, from devices such as multimode phased array radars [5] to entire battle management systems [6]. Such systems are fundamentally distinguished by the fact that they operate in environments with dynamically uncertain properties. These uncertainties include transient and sustained resource overloads due to context-dependent activity execution time demands and arbitrary activity arrival patterns. Nevertheless, such systems' desire the strongest possible assurances on activity timeliness behavior.

When resource overloads occur, some activities must be scheduled to complete at suboptimal times or be rejected. The criteria used for scheduling should include the activities' urgency and importance, which are orthogonal. In some overload cases, the objective is to complete the most

1. Distributable threads first appeared in the Alpha OS [3] and later in Alpha's descendant, the MK7.3 OS [4].

- *K. Channakeshava and B. Ravindran are with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061. E-mail: {kchannak, binoy}@vt.edu.*
- *E.D. Jensen is with the Information Technologies Directorate, The MITRE Corporation, Bedford, MA 01730. E-mail: jensen@mitre.org.*

Manuscript received 17 Aug 2004; revised 3 Aug. 2005; accepted 24 Aug. 2005; published online 22 Feb. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0271-0804.

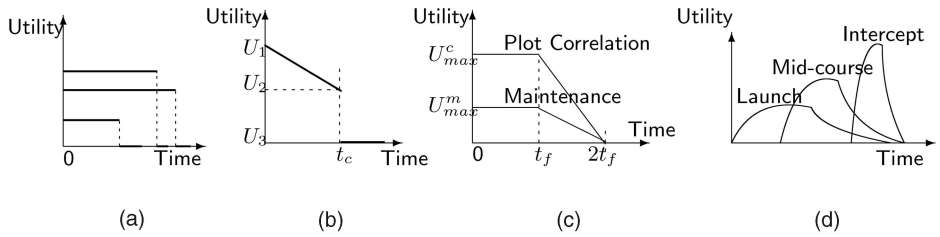


Fig. 1. Example time constraints specified using time/utility functions: (a) Step TUFs, (b) MITRE/TOG AWACS track association TUF [9], and (c), (d) GD/CMU Coastal Air Defense System TUFs [10].

important of the most urgent activities and, in other cases, it is to complete the most urgent of the most important activities. Thus, a scheduling facility that accommodates overloads must provide a way of spcifying trade-offs between urgency and importance. When the resources are underloaded, this distinction is not needed because, if urgency alone is the optimality criterion, such as with the EDF discipline, all deadlines are guaranteed to be met [7].

Deadlines by themselves cannot express both urgency and importance. Thus, we consider the abstraction of time/utility functions (or TUFs) [8] that express the utility of completing an application activity as a function of that activity's completion time. We specify deadline as a binary-valued, downward "step" shaped TUF; Fig. 1a shows examples. Note that a TUF decouples importance and urgency—i.e., urgency is measured on the X-axis, and importance is denoted by utility on the Y-axis.

Besides overloads, many dynamic real-time systems are also distinguished by activities that are subject to *nondeadline* time constraints, such as those where the utility attained for activity completion *varies* (e.g., decreases, increases) with completion time. This is in contrast to deadlines where a positive utility is obtained for completing the activity anytime before the deadline, after which zero or arbitrarily negative utility is obtained. Fig. 1b, Fig. 1c, and Fig. 1d show example time constraints from two real applications in the defense domain (see [9] and references therein for application details).

When activity time constraints are specified using TUFs, which subsume deadlines, the scheduling criteria are based on accrued utility, such as maximizing the sum of the activities' attained utilities. Such criteria are called Utility Accrual (or UA) criteria and scheduling algorithms using such criteria are called UA scheduling algorithms. Several UA scheduling algorithms are presented in the literature [11], [12], [13], [14]. RTC2 has IDL interfaces for the UA scheduling discipline, besides others such as fixed priority, EDF, and LLF.

UA algorithms that seek maximal summed utility under downward step TUFs (or deadlines) [12], [13], [15] default to EDF during underloads since EDF can satisfy all deadlines during those situations. Consequently, they obtain the maximum total utility during underloads. When overloads occur, they favor activities that are more important (since more utility can be attained from them), irrespective of their urgency. Thus, UA algorithms' timeliness behavior subsume the optimal timeliness behavior of deadline scheduling.

1.2 UA Channel Establishment

When a multihop network—i.e., one where end-hosts are interconnected by multiple switches and routers—is considered as the underlying platform for an RTC2 application, distributable threads will compete for node-local resources as well as network resources. Node-local resources are those resources that are local to a system "node" such as an end-host or a router. Such resources include physical resources (e.g., processor, disk, I/O) and logical resources (e.g., locks).

Network resources may include *real-time channels*. Traditionally, a real-time channel is a unidirectional virtual circuit that is established for application-level messages in a multihop network with guaranteed timeliness properties [16]. In the context of an RTC2 application, application-level messages include those that are generated when threads invoke operations on remote objects and thus transit nodes. Thus, such messages contend for real-time channels in multihop networks. Moreover, they are indirectly subject to the timeliness properties of their "parent" threads, on which time constraints and timeliness optimality criteria are explicitly expressed.

While scheduling of threads on nodes and resolution of node-local resource contention among threads is performed by a scheduling algorithm, real-time channels are established by a channel establishment algorithm. Thus, when threads are subject to TUF time constraints and UA optimality criteria, UA scheduling algorithms and UA channel establishment algorithms must be used for coherent system-wide resource management and for improved timeliness optimization.

In this paper, we consider the problem of UA channel establishment in multihop networks. We consider thread time constraints that are specified using TUFs and the optimality criterion of maximizing the sum of threads' attained utilities. We focus on messages which underlie threads, which are indirectly subject to the thread TUF time constraints. For the purpose of maximizing the sum of threads' attained utilities, we consider the problem of establishing channels for the messages such that the sum of *messages'* attained utilities are maximized.

Note that timeliness optimization at the message-level is completely consistent with RTC2's Case 2 approach as thread messages, on behalf of threads, contend for real-time channel resources. The contention among these messages is resolved using the thread scheduling parameters (that messages carry) and considering the thread-level optimality criterion at the

message-level. Thus, message-level timeliness optimization contributes to thread-level timeliness optimization.

The UA channel establishment problem can be shown to be $\mathcal{NP}$-complete.[2] Thus, we present 1) a single node algorithm called Local Decision for Utility accrual Channel Establishment (or LocDUCE) and 2) a distributed algorithm called Global Decision for Utility accrual Channel Establishment (or GloDUCE). The algorithms heuristically compute channels, seeking to maximize the sum of messages' attained utilities, as much as possible. We evaluate the algorithms by simulation studies and implementation measurements, and by comparing them with the Internet standard Open Shortest Path First (OSPF) routing algorithm. Our simulation studies and implementation measurements reveal that LocDUCE and GloDUCE accrue significantly higher utility than OSPF. Furthermore, we observe that GloDUCE performs better than LocDUCE under high downstream link traffic conditions.

We also compare our algorithms with the optimal algorithm for small problem sizes. Our comparisons with the optimal algorithm reveal that both LocDUCE and GloDUCE perform very close to the optimal for some homogeneous TUF shapes (above 90 percent of the optimal). The performance of LocDUCE and GloDUCE degrades for heterogeneous TUFs (to about 88 percent). We also evaluate the algorithms' performance for varying upstream and downstream traffic. In both cases, we observe that the performance of LocDUCE degrades to some extent (as expected), while GloDUCE yields close to the optimal performance. Further, we observe that performance of GloDUCE degrades to 85 percent of the optimal for heterogeneous TUFs.

Thus, the contribution of the paper is the LocDUCE and GloDUCE utility accrual channel establishment algorithms. Most of the past efforts on real-time channel establishment [16], [18], [19], [20] focus on the deadline time constraint. To the best of our knowledge, we are not aware of any other efforts that solve the problem of TUF/UA channel establishment, which is solved by the LocDUCE and GloDUCE algorithms.

1.3 Paper Outline

The rest of the paper is organized as follows: We describe our message, timeliness, and system models in Section 2. Section 3 presents a "bottom-up" description of the LocDUCE and GloDUCE algorithms. We discuss the simulation study in Section 4. Section 5 discusses the implementation of the algorithms and the resulting performance measurements. We overview past research on real-time channel establishment and contrast them with our work in Section 6. Finally, we conclude the paper and identify future work in Section 7.

2. To maximize the system-wide accrued utility (under nonincreasing TUFs), it is necessary to find the lowest delay path from a source to its destination. For this, each intermediate router must compute a minimum cost tree with end-to-end delay constraints. It has been shown in [17] that this problem is $\mathcal{NP}$-complete.

2 The Message, Timeliness, and System Models

We consider internode messages that are generated when distributable threads invoke operations on remote objects (and thus transcend node boundaries). Thus, all messages are assumed to be created as a result of the threads' remote operation invocations. Table 1 shows the notations employed in the paper. We denote the set of messages as $m_i \in M, i \in [1, n]$. The arrival instant and termination time (or deadline) of a message m_i are denoted as $A(m_i)$ and $X(m_i)$, respectively. A message's termination time is the time after which its utility is zero (see further on message TUFs). The bit length of a message m_i at the data link-layer is denoted as $b(m_i)$. The physical framing overheads increase this size into an actual bit length $b^{'}(m_i) > b(m_i)$ on the wire. Thus, the transmission latency of a message m_i is given by $l_i = b^{'}(m_i)/\psi$, where ψ denotes the nominal throughput of the underlying network, e.g., 10^9 bits/s for Gigabit Ethernet.

We consider the *unimodal arbitrary arrival model* for messages, as this model dominates other arrival models, including the aperiodic, sporadic, and periodic arrival models, due to the "strength" of the "adversary" embodied in the model [21]. For a message m_i, the unimodal arrival model defines the size of a sliding time window $w(m_i)$ and the maximum number of arrivals $a(m_i)$ that can occur during that window.

Each message $m_i \in M$ has a time constraint that is expressed using a TUF. The TUF time constraint of a message is derived from the time constraint of the thread to which the message belongs. We denote message m_i's TUF as $U_i(.)$. Thus, m_i's arrival at its destination host application layer (which triggers the invoked operation on an object on the host) at a time t will yield a utility $U_i(t)$. Though TUFs can take arbitrary shapes, we restrict our focus to monotonically nonincreasing (*unimodal*) TUFs. An example can be seen in Fig. 1b.

We assume that $U_i(t) \geq 0, \forall t \in [A(m_i), X(m_i)], i \in [1, n]$.

We consider the system to be a multihop network with an arbitrary topology that consists of several networks of various types. A typical example is a local area network (or LAN) that interconnects a set of hosts using one or more switches. A collection of such LANs can further be interconnected together by a set of routers, thus forming a wide area network (or WAN). In such a network, a message can travel from a source host to a destination host by passing through a number of intermediate nodes (switches and routers). The route from a source to a destination thus includes a set of nodes where a message can be stored and forwarded to the appropriate next hop. Fig. 2a shows an example of such a network.

These types of distributed (real-time) systems that we focus on in this work are purely *intra-enterprise*—i.e., they are deployed within an enterprise with some level of system-wide agreement on infrastructure technology, such as agreement on timeliness semantics and sufficiently synchronized clocks. We consider such a WAN as our target system, which is consistent with the current deployment of distributed real-time systems (e.g., Real-Time CORBA systems). It should be noted that it is extremely difficult to create the

TABLE 1
Notations

p,q	application packets
c	clock synchronization packet
m_i	a message i
$A(m_i)$	message creation or arrival instant of m_i
$X(m_i)$	termination-time of message m_i
$b'(m_i)(b(m_i))$	bit length of message m_i with(without) headers in bytes
ψ	nominal network throughput in bps
l_i	transmission latency of message m_i in seconds ($l_i = b^{'}(m_i)/\psi$)
$w(m_i)$,$a(m_i)$	window of time and number of arrivals in that window
$U_i(t)$	message utility at time t
α	size of set $\mathcal{A}$ ($\alpha = \|\mathcal{A}\|$)
δ	aggregate worst case execution, dispatching and queue transfer time
θ	clock synchronization packet period
M	set of messages $m_i \in M, i \in [1,n]$
P_i	upper bound on the number of packets that can arrive at the output queue
$\mathcal{A}$	set of messages in a queue ($\mathcal{A} \subseteq M$)
$\mathcal{S}(\mathcal{A})$	set of all message arrangement sequences
σ	complete sequence of messages in a queue ($\sigma \in \mathcal{S}(\mathcal{A})$)
π	partial sequence of messages in a queue ($\pi \subseteq \sigma$)
$I(p)$	interference interval for packet p i.e. $I(p) = [(A(p) - X(q), (A(p) + X(p)))]$
$O_1^i(p)$,$O_0^i(p)$	delay upper bound at first and second output queues respectively
$R_1^i(p)$	arrival rate at second output queue
$N_j(p)$	node delay for packet p
$\Gamma(p,t)$	pseudo-slope of packet p at time t
$\Delta_{q,p}$	difference in utility when packet q is transmitted before p
$\hat{N}_j(p)$	delay at destination node j
C_i^j	channel from source node i to destination node j
$R(p, C_i^j)$	end-to-end delay for packet p in channel C_i^j

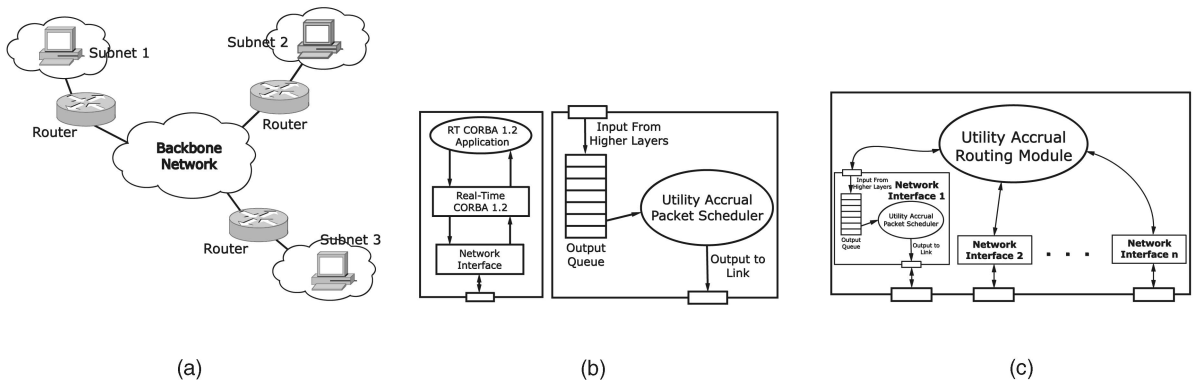


Fig. 2. The system model and its components. (a) A multihop network. (b) End-host and interface structure. (c) Router system structure.

technical agreement and coordination between multiple enterprises which is required for an inter-enterprise deployment—we exclude such systems in our work.

Fig. 2b and Fig. 2c show the components used in the system model. Fig. 2b shows the end-host and interface structures and Fig. 2c shows the router structure. We assume that each system node—end-hosts, switches, and routers—is equipped with the UPA (utility accrual packet scheduling algorithm) presented in [14]. The messages arrive at the MAC layer and are placed on the output queue for the outgoing network link. When the link becomes physically free for transmission, the (UPA) message scheduler selects a message from the queue for transmission toward an intermediate node, such as a router or a gateway.

In the rest of the paper, we will refer to message and packet interchangeably and a message is assumed to be transported as a single packet.

3 The LocDUCE and GloDUCE Algorithms

We follow a "bottom-up" approach in describing LocDUCE and GloDUCE. The key step in both the algorithms is to estimate the delay incurred by a message for a single hop under UPA. Thus, we first overview UPA, analyze the

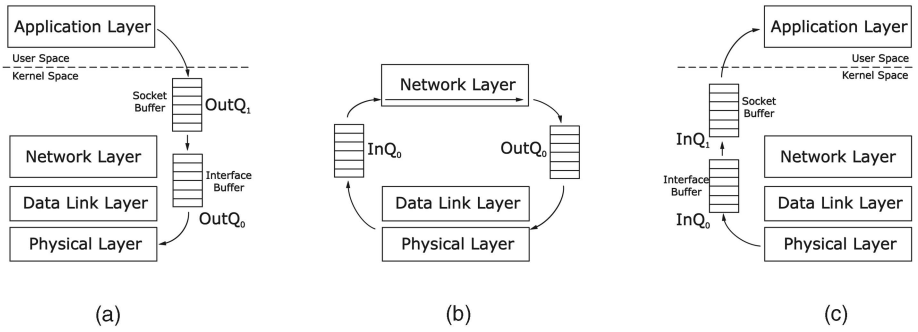


Fig. 3. System model and input/ouput queues at source, router, and destination nodes. (a) Source node. (b) Router. (c) Destination node.

single hop delay under UPA, and, subsequently, present the LocDUCE algorithm. Detailed experimental evaluation of the LocDUCE algorithm is presented in [22]. Since GloDUCE is a distributed algorithm, it estimates the path delay from source to destination under UPA. Thus, we analyze the path delay before presenting GloDUCE.

3.1 Overview of UPA

UPA is a message scheduling algorithm present at the MAC layer of system nodes (e.g., hosts, switches) for selecting outbound messages for transmission. The algorithm maximizes the sum of messages' attained utilities. UPA first constructs a tentative schedule by sorting messages in decreasing order of their "return on investments." The return on investment for a message is the potential utility that can be obtained by spending a unit of network transmission time for the message. It is determined as the ratio of the maximum possible message utility to the message termination time. In [14], this ratio is called "pseudoslope," since it only approximates the slope.

Messages that are found to be infeasible are dropped from the tentative schedule. The algorithm then maximizes the *local* aggregate utility for a particular sequence of messages. Consider two schedules, $\sigma_a = \langle \pi_1, m_i, m_j, \pi_2 \rangle$ and $\sigma_b = \langle \pi_1, m_j, m_i, \pi_2 \rangle$, of a message set $\mathcal{A}$ (see Table 1 for a description of the notations). The only difference between the schedules is the order in which messages m_i and m_j appear in them. The scheduling decision at time t, where $t = \sum_{k \in \pi_1} l_k$, that will lead to maximum local aggregate utility is determined by computing:

$$\Delta_{i,j}(t) = \big[U_i(t + l_i) + U_j\big(t + l_i + l_j\big)\big] - \big[U_j\big(t + l_j\big) + U_i\big(t + l_j + l_i\big)\big].$$

Thus, if $\Delta_{i,j}(t) \geq 0$, σ_a will yield a higher aggregate utility than σ_b.

UPA maximizes local aggregate utility by examining adjacent pairs of messages in the schedule, computing Δ, and swapping the messages, if the reverse order can lead to higher local aggregate utility. The procedure is repeated until no swaps are required. The message that appears first in the resulting schedule is then selected for transmission. Thus, given a schedule consisting of α messages, UPA's objective is to $Maximize \sum_{i=1}^{\alpha} U_i(t_i)$, where t_i is the absolute time at which message m_i arrives at its destination. This scheduling problem is $\mathcal{NP}$-hard [14]. It has been shown in [14] that UPA is the best heuristic algorithm for this problem, outperforming the previously known best algorithm presented in [23]. UPA is an asynchronous algorithm in the sense that it is invoked whenever the outgoing network link becomes "free" for transmission. In the implementation of UPA as presented in [14], this is accomplished by having the network device driver interrupt UPA when the outgoing link becomes free.

3.2 Single Hop Delay

Fig. 3a, Fig. 3b, and Fig. 3c show the typical set of input and output queues through which messages flow from the application layer at a source node, via an intermediate node (switch and/or router), to the application layer at a destination node. The delay incurred by the message in all the queues has to be taken into account while calculating the end-to-end delay. Note that, in our system model, all packet queues are assumed to be scheduled by UPA.

3.2.1 Delay at First Output Queue

Consider an application packet p that arrives at the first output queue of a source node i (denoted $OutQ_1$ in Fig. 3a). In [24], we show that the upper bound on the total delay incurred by p in this queue is given by:

$$O_1^i(p) = \sum_{q \in P_i} \left\lceil \frac{X(p) + X(q)}{w(q)} \right\rceil a(q)\delta + \left\lceil \frac{X(p)}{\theta} \right\rceil \delta. \qquad (1)$$

For the definitions of terms used in (1), see Table 1.

This upper bound $O_1^i(p)$ derived in [24] is not *tight*. This is because, $O_1^i(p)$ is established in [24] by observing that any packet q will be scheduled by UPA before packet p, only if q arrives no sooner than $A(p) - X(q)$ and no later than $A(p) + X(p)$, where $A(p)$ and $X(p)$ denote p's arrival time at the output queue and termination time, respectively. This interference interval for p, $I(p) = [A(p) - X(q), A(p) + X(p)]$ is only sufficient for a packet q to interfere with the transmission of packet p. Packet q can interfere with packet p only if q arrives during this interval.

Developing a necessary and sufficient interference interval will require schedule construction. To avoid this and still obtain a "tighter" interference interval, we consider the notion of an interference interval during which a packet q may interfere with packet p with a very high likelihood.

Observe that UPA sorts packets by decreasing order of their pseudoslopes. Thus, if the pseudoslope of packet q is larger than that of packet p, for all times during p's interference interval $I(p)$, then q has a very high likelihood

for interfering with p's transmission. Thus, if $\Gamma(p,t)$ denotes the pseudoslope of packet p at time t, then our first tighter interference condition becomes:

$$\Gamma(q,t) > \Gamma(p,t), \forall t \in I(p). \tag{2}$$

Another key step in UPA's scheduling process is examining adjacent pairs of packets in the schedule, computing Δ, and swapping the packets if the reverse order can lead to higher local aggregate utility. Suppose $u_{p,q}(t)$ is the utility obtained by transmitting p before q, and $u_{q,p}(t)$ for transmitting q before p, then $\Delta_{q,p}(t) = u_{q,p}(t) - u_{p,q}(t)$. Thus, if $\Delta_{q,p}(t) \geq 0$ for all time instants t during the interval $I(p)$, then, again, packet q has a very high likelihood for interfering with p's transmission. This becomes our second tighter interference condition:

$$\Delta_{q,p}(t) \geq 0, \forall t \in I(p). \tag{3}$$

We thus determine a *tighter* upper bound on p's delay in the first output queue at a source node i using (1) by considering all packets $q \in P_i$, only if q satisfies (2) and (3).

3.2.2 Delay at Second Output Queue

The delay incurred by a packet at the second output queue (denoted $OutQ_0$ in Fig. 3a) can be similarly derived, except for two issues: 1) The input arrival rate of packets into the second output queue will now change as it will depend upon the rate at which packets will be output from the first output queue, and 2) packet transmission times on the outgoing network link (from the source node to the next intermediate node).

For a packet p that can arrive at the first output queue of a source node i for a maximum of $a(p)$ times during an interval $w(p)$, the rate at which the packet will be output from the queue is given by:

$$R_1^i(p) = \left\lceil \frac{O_1^i(p)}{w(p)} \right\rceil a(p). \tag{4}$$

This will be the rate at which p will arrive at the second output queue.

For a clock synchronization packet c, the arrival rate at the second output queue is given by:

$$R_1^i(c) = \left\lceil \frac{O_1^i(c)}{\theta} \right\rceil. \tag{5}$$

Thus, an upper bound on the delay incurred by a packet p to arrive at the next intermediate node since its arrival at the source node's second output queue is given as:

$$\begin{aligned} O_0^i(p) = \sum_{q \in P_i} & \left[X(p) + X(q) - \frac{b'(p)}{\psi} \right] R_1^i(q) \left[\frac{b'(q)}{\psi} + \delta \right] \\ & + \left[X(p) - \frac{b'(p)}{\psi} + \frac{b'_c}{\psi} \right] R_1^i(c) \left[\frac{b'_c}{\psi} + \delta \right]. \end{aligned} \tag{6}$$

3.2.3 Total Delay at a Node

Thus, an upper bound on the total delay incurred by a packet p to arrive at the next intermediate node j, since its arrival at a source node i's first output queue, is given as:

$$N_i(p) = O_1^i(p) + O_0^i(p). \tag{7}$$

Observe that the upper bound on the total delay incurred by a packet p to arrive at the first input queue (denoted by InQ_0 in Fig. 3b) of any node j is also given by (6). Further, node j can either be an intermediate or a destination node. This is because the analysis for queues at the source node (denoted $OutQ_1$ and $OutQ_0$ in Fig. 3a) also holds for the queues at an intermediate node (denoted InQ_0 and $OutQ_0$ in Fig. 3b). Note that we have considered all interfering packets as $q \in P_i$. This includes packets generated by the same thread as well as other threads.

3.3 LocDUCE Algorithm

The key heuristic employed by the algorithm is to allocate channels for messages in decreasing order of their potential return on investments. The return on investment for a message is simply the timeliness utility that can be obtained when the message is delivered to its destination.

To estimate the maximum possible return on investment from a message, LocDUCE first allocates channels to each message, assuming that the messages will not interfere with each other (i.e., under zero contention between messages). Thus, the algorithm considers the network as an unweighted graph, where each vertex represents a system node and an edge represents a connection between a pair of nodes. For each message, LocDUCE determines the shortest path-distance from the source to destination (i.e., the one with the smallest hop-count) by running the breadth-first-search (BFS) algorithm. The path (or channel) with the shortest path-distance for a message will yield the maximum possible utility for the message since all TUFs considered are *nonincreasing*. Note that this utility is the theoretical maximum possible utility; it may not be achievable in practice due to message contention.

LocDUCE computes channels for messages in decreasing order of the maximum possible message utility. Consider the allocation of the channel for the kth message m_k. LocDUCE includes messages m_1 to m_{k-1} in P_i, where P_i is a set of messages that arrive at node i and are transmitted to the next-hop node $(i+1)$. For message m_k (in decreasing maximum possible utility order), the algorithm at any node i first updates the network graph such that the edge connecting i to the next-hop node $(i+1)$ is annotated with the interference m_k may suffer because of messages m_1 to m_{k-1}. Once P_i has been updated with m_{k-1} messages, any message that is included on the edge between nodes i and $(i+1)$ may suffer interference from the m_{k-1} previously allocated messages.

LocDUCE uses Dijkstra's shortest-path algorithm to determine the shortest path channel for message m_k. Again, the shortest path will yield the largest possible maximum utility for the message. For determining the shortest path, the weight of the outgoing edge from any node i to a next-hop node $(i+1)$ in m_{k-1}'s channel is determined using (7). Note that (7) gives an upper bound on the total delay incurred by the message to arrive at the next intermediate node by traveling through the edge. The algorithm repeats the process for each message $m_i \in M, i \in [1,n]$.

While LocDUCE computes the channel for message m_k, it does not consider the interference that m_k can cause on

messages $m_i, i \in [1, k-1]$ and accordingly update their channels that were computed in earlier steps. This is precisely due to the algorithm's heuristic nature. LocDUCE ignores such "backward" interference and reasons that it will not be significant (as channels are allocated in the decreasing order of maximum possible message utility). Further, correcting such backward interference will be computationally expensive. A high level description of LocDUCE is given in Algorithm 1.

Algorithm 1: Overview of LocDUCE

```
1  graph: Network Graph; t_trans: Transmission time;
2  P_k: set of messages ordered by "Return of Investment";
3  for m_i ∈ M, i ∈ [1, n] do
4      /* For Node k */
5      if m_i ∉ P_k then
6          hops := BFS (graph, m_i);
7          t_d := t_cur + (hops × t_trans);
8          /* Calculate return of investment (roi) */
9          roi := U_i(t_d);
10         P_k[j] ← (m_i, roi) : (P_k[j].roi > P_k[j+1].roi);
11         /* Update network graph with m_{k-1} */
12         for ∀p ∈ P_k, p ∈ [1, (j-1)] do
13             UpdateGraph (graph, P_k[p]);
14         for ∀p ∈ P_k, p ∈ [j, ···] do
15             P_k[p].nextHop := Dijkstra (graph, P_k[p].msg);
16             UpdateGraph (graph, P_k[p]);
17     else
18         nextHop := P_k[m_i].nextHop;
```

3.4 Determining End-to-End Delay

A channel from source node i to destination node j is a sequence $C_i^j = \langle n_0, n_1, n_2, \cdots, n_m \rangle$ of m hops, where n_0 represents node i and n_m represents node j. To determine an upper bound on the end-to-end delay incurred by a packet on a channel (from source application layer to destination application layer), we need to aggregate the delay incurred by the packet at each node in the channel. Thus, an (optimistic) upper bound on the total delay incurred by a packet p on a channel C_i^j of m hops is given by:

$$R\Big(p, C_i^j\Big) = \left[\sum_{i=1}^{m} N_i(p)\right] + \hat{N}_j(p), \tag{8}$$

where $\hat{N}_j(p)$ denotes an upper bound on the total delay incurred by packet p to arrive at the application layer of the destination node j, since its arrival at InQ_0 of j. $\hat{N}_j(p)$ is determined similar to (7), except that the packet transmission times on the outgoing network link considered in (6) are excluded.

3.5 GloDUCE Algorithm

GloDUCE has the same heuristic (as LocDUCE) of allocating channels for messages in decreasing order of their potential return on investments. Similar to LocDUCE, GloDUCE first allocates channels for each message, assuming that the messages will not interfere with each other. For each message, GloDUCE determines the shortest path from the source to destination (i.e., the one with the smallest hop-count) by running BFS.

GloDUCE differs from LocDUCE in the following way: While the channel for the kth message m_k is determined, the algorithm at any node first updates the network graph with the channel of the $(k-1)$th message, denoted m_{k-1}, computed in the previous step, i.e., for each node i, that lie in message m_{k-1}'s channel, GloDUCE includes m_{k-1} to the set P_i. Here, P_i is the set of messages that have outgoing channels from node i and can potentially interfere with message m_k. This is achieved through broadcasting the channel information after establishment. Once P_i has been updated with m_{k-1}'s channel, any channel that includes the outgoing edge from node i in m_{k-1}'s channel may suffer interference from m_{k-1}. Such channels are updated and reestablished.

GloDUCE uses Dijkstra's algorithm to determine the shortest path for message m_k. Again, the shortest path will yield the maximum possible utility for the message. For determining the shortest path, the weight of the outgoing edge from any node to a next-hop node in m_{k-1}'s channel is determined using (7). Note that (7) gives an upper bound on the total delay incurred by the message to arrive at the next intermediate node by traveling through the edge. The interference along the entire channel is determined using (8).

When the GloDUCE algorithm on any node receives an update from another node, it determines whether the new channel will cause interference to an already established channel. If an already established channel is affected by any new channel, the algorithm recalculates the channel. A high level description of GloDUCE is given in Algorithm 2.

Algorithm 2: Overview of GloDUCE

```
1  graph: Network Graph; t_trans: Transmission time;
2  P: set of all messages in the network ordered by "Return of Investment";
3
4  for m_i ∈ M, i ∈ [1, n] do
5      /* For any Node */
6      if m_i ∉ P then
7          hops := BFS (graph, m_i);
8          t_d := t_cur + (hops × t_trans);
9          /* Calculate return of investment (roi) */
10         roi := U_i(t_d);
11         P[j] ← (m_i, roi) : (P[j].roi > P[j+1].roi);
12         /* Update network graph with m_{k-1}s */
13         for ∀p ∈ P, p ∈ [1, (j-1)] do
14             UpdateGraph (graph, P[p]);
15         for ∀p ∈ P, p ∈ [j, ···] do
16             P[p].nextHop :=
                   Dijkstra (graph, P[p].msg);
17             UpdateGraph (graph, P[p]);
18             BroadcastUpdate (P[p]);
19     else
20         nextHop := P[m_i].nextHop;
```

4 Simulation Study

To study the performance of the algorithms in large data spaces, we randomly generate network topologies with increasing edge-to-node ratios. This increases the network connectivity and provides a basis for comparison of the proposed algorithms and OSPF. Some thread characteristics, like the shape of the TUFs, message sizes, and message termination times, are varied during the study. Table 2 shows the parameters of the simulation experiments. A more detailed description of the simulation model and tools employed is available in [24].

TABLE 2
Simulation Study Parameters

Message Properties	Parameters
Inter-arrival Time (seconds) Mean – 0.25 Variance – 0.0115	$Constant(0.25)$, $Uniform(0.0642, 0.44)$, $Normal(0.25, 0.0115)$, $Max(0.065, Exp(0.25))$, $Pareto(3.54, 0.18)$
Message Length (Bytes) Mean – 2000 Variance – 588	$Constant(2000)$, $Uniform(1958, 2042)$, $Normal(2000, 24.24)$, $Max(800, Exp(2000))$, $Pareto(83.49, 1976.04)$
Message Termn. Time (seconds) Mean – 1.5 Variance – 0.20	$Constant(1.5)$, $Uniform(0.73, 2.27)$, $Normal(1.5, 0.45)$, $Max(0.75, Exp(1.5))$, $Pareto(4.5, 1.17)$
System Properties	**Parameters**
Edge/Node Ratio	0.933–1.867

The topology used in the simulation experiments consisted of 15 routers interconnecting five source-destination pairs. The links connecting the nodes had a bandwidth of $64kbps$. The values of the bandwidth and data rate of traffic generators are chosen such that the links are loaded when more than one *flow*[3] uses the same link. Each message generated by a particular flow has the same message characteristics (such as TUF, message size, etc.). The simulation experiments can be broadly classified into two categories:

1. Homogeneous TUFs—All threads have a similar type of TUF.
2. Heterogeneous TUFs—Each thread has a different type of TUF.

Experiments were conducted by varying interarrival times, message sizes, and message termination times under these classes of TUF shapes. The experiments were repeated for different edge to node ratios from 0.933 to 1.867. Each simulation experiment was repeated for different seed values and the average and standard deviation were calculated. Here, we present the results that were obtained for the Pareto distribution; results under other distributions can be found in [24].

4.1 Homogeneous TUFs

Under the homogeneous case, the simulation was conducted with all threads having the same TUF shape including `step` TUF (Fig. 4a), `soft-step` TUF (Fig. 4b), and track association (`asoc`) TUF (Fig. 4c). The actual TUF characteristics for each thread are outlined in Table 3. In addition, the termination times for all threads is set as 3.0 seconds.

4.1.1 Varying Message Interarrival Times

In Fig. 5, we plot the system-wide percentage utility accrued (or % UA) and percentage termination time misses (or % TM) with respect to the edge-to-node ratio for OSPF, LocDUCE, and GloDUCE. The edge-to-node ratios are indicated by `r0`, `r1`, etc. The results for other TUFs follow the same trend and have been omitted because of page constraints; they can be found in [24].

3. A flow here is the aggregation of the remote calls made by a single thread over the duration of the simulation.

The utility obtained under OSPF is not affected by increase in the connectivity. This shows that, even with an increase in the edge-to-node ratio, there is no noticeable change in the shortest paths. Hence, the system-wide utility obtained remains low and the termination time misses remains high. This condition offers a very good scenario for comparing performance with the LocDUCE and GloDUCE algorithms. From the plot, we find that both LocDUCE and GloDUCE have a similar performance except for `r4`. LocDUCE has a lower system-wide utility than GloDUCE for this ratio. To investigate this further, Fig. 6 shows the % TM values obtained for the threads with LocDUCE and GloDUCE at `r4`. In this plot, the threads are denoted as `T1`, `T2`, etc. We observe that the % TM for `T2` under GloDUCE is less when compared to LocDUCE (Fig. 6).

Fig. 7 shows the partial topology for `r4` along with the routes taken for `T2`, `T4`, and `T5`. The paths taken by the other threads are not shown here as they do not affect `T2`. Also, from Table 3, we find that `T2` has lesser utility than `T4` and `T5`. The routes shown for messages of `T2`, `T4` and `T5` in Fig. 7a and Fig. 7b correspond to the paths taken under LocDUCE and GloDUCE, respectively.

When LocDUCE is used for channel establishment, router `R2` routes `T4` along the path `R2-R0-R3`. The UPA

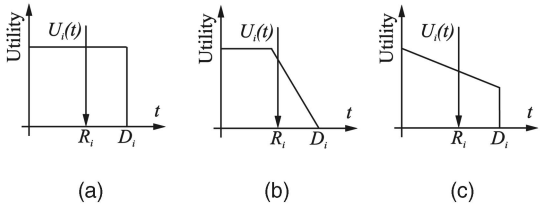
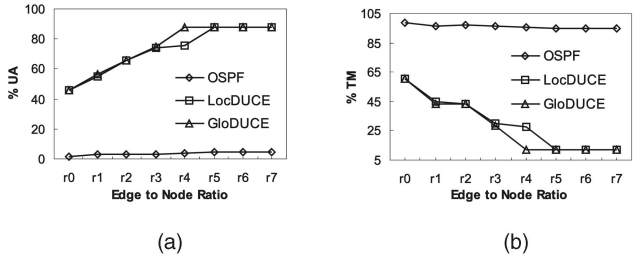


Fig. 4. TUFs for homogeneous class of simulation runs. (a) Step. (b) Soft-step. (c) (MITRE/TOG AWACS) Track association.

TABLE 3
Maximum TUF Utility of Threads

Thread	Max Utility
Thread-1	1.5
Thread-2	2.0
Thread-3	2.5
Thread-4	3.0
Thread-5	3.5



(a) (b)

Fig. 5. Performance comparison under varying interarrival times (`Step` TUFs). (a) % UA. (b) % TM.

scheduler of the link `R0-R3` prefers the higher utility traffic of `T4` over that of `T2`. Thus, `T2` suffers very high % TM under LocDUCE. The same topology under GloDUCE, yields a different result, as shown in Fig. 7b. Each router maintains information about channels established through other routers and, hence, avoids the paths that are loaded. In this example, `R0` routes the traffic from `T2` along a longer path as it is aware of the fact that the link `R0-R3` is in the channel of `T4`. Thus, GloDUCE is able to achieve a higher % UA and lower % TM (as in Fig. 6) for `T2`.

4.1.2 Varying Message Termination Time

In this simulation study, the messages have varying termination time values. The termination times of the messages were varied around a mean value and the performance of the algorithms was measured. The performance is as shown in Fig. 8. Here, even though we observe the same trend as before, i.e., both LocDUCE and GloDUCE outperform OSPF, the magnitude of the differences is reduced considerably. This can be attributed to the fact that the termination time values are randomly chosen in the different distributions considered and affect all the algorithms in a similar manner. LocDUCE and GloDUCE perform better than OSPF even under this condition. In some cases, GloDUCE performs slightly better than LocDUCE.

4.2 Heterogeneous TUFs

In the heterogeneous case, the simulation comparison was conducted with threads having different TUFs. For example, threads with `step`, `soft-step`, and `linear` TUFs were all simultaneously executed. Both maximum utility value and TUF type were varied. The experiments were performed with threads characterized by TUFs shown in Fig. 9, in addition to the ones in Fig. 4.

In order to offset the effect of the location of the threads on a particular source-destination pair, simulation runs were conducted by rotating the threads so that the thread

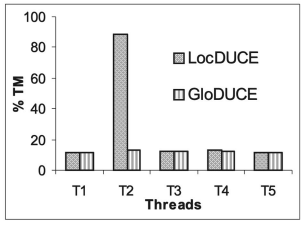
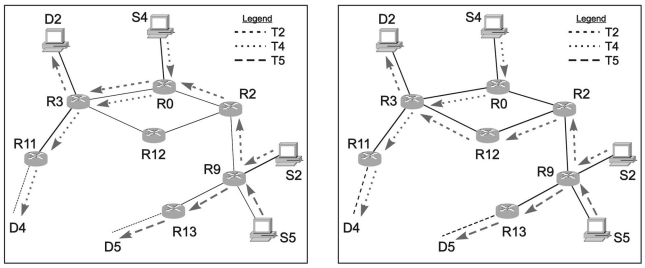


Fig. 6. Percentage TM Comparison for Ratio `r4`.



(a) (b)

Fig. 7. Partial topology for edge to node ration `r4`. (a) LocDUCE. (b) GloDUCE.

with a certain TUF executed on all the possible source nodes. For example, `T1` with a `step` TUF executes from source `S1` in one run, from `S2` in another, and so on, until it is run between all possible source-destination pairs. The results presented here are the average of all such runs. Table 4 shows the shape of the TUF and the maximum utility of the threads. The termination times of all the threads is set as 3.0 seconds.

4.2.1 Varying Message Interarrival Times

The same set of simulation experiments outlined in Section 4.1.1 was repeated with threads having time constraints, as listed in Table 4. Fig. 10 shows the % UA comparison of OSPF, LocDUCE, and GloDUCE when the interarrival times are varied and TUFs are heterogeneous.

In Fig. 11, we show the comparison of the utility obtained for various TUFs between LocDUCE and GloDUCE for the ratio `r4`. For this topology, GloDUCE performs better than LocDUCE, which shows the difference between the two algorithms. In the case of ratio `r4`, GloDUCE prefers the `step` and `soft-step` TUFs over the `linear` TUF and achieves a better system-wide utility than LocDUCE.

The results for heterogeneous TUFs where the message sizes and termination times are varied follow the same trend as observed for the homogeneous TUFs. These results can be found in [24].

4.3 Downstream Loading

To observe the difference between LocDUCE and GloDUCE, a specific simulation was setup. Fig. 12 shows the topology and link bandwidth considered for performing this study. In this experiment, there are three source nodes generating traffic. Sources `S1` (thread `T1`) and `S2` (thread `T2`) generate the thread traffic under observation. Source `S3` (thread `T3`) generates an increasing rate of background traffic to load

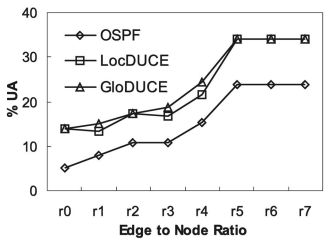


Fig. 8. Varying message termination times: % UA comparison

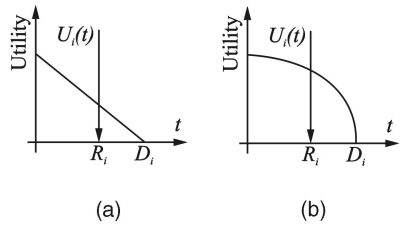


Fig. 9. TUFs for heterogeneous experiments.

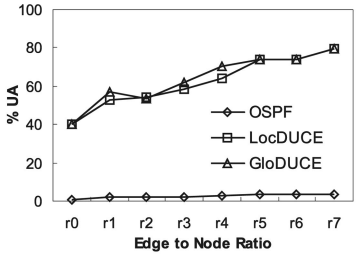


Fig. 10. Percentage UA comparison for varying interarrival times under heterogeneous TUFs.

the link R1-R2. The destination nodes D1, D2, and D3 form the sink for the messages. Table 5 shows the characteristics of the threads used in the experiment. Table 5 also lists the data rate of the threads. In the experiments, all threads have Step TUFs and the same termination time value.

Fig. 13 shows the % UA and % TM of T2 for LocDUCE and GloDUCE algorithms. The background rate in Fig. 13 is the rate at which T3 generates traffic. From these plots, we find that LocDUCE performs better than GloDUCE at very low background rates and, for higher background rates, GloDUCE outperforms LocDUCE.

GloDUCE performs better than LocDUCE because of its distributed nature. Assuming that T1 transmits first, GloDUCE routes its traffic along the path R0-R2, while T2 routes along the path R0-R1-R2. When T3 starts transmitting and, because it has a higher utility than T2, R1 chooses the shortest available path along R1-R2 and creates a channel for T3. R1 broadcasts this channel information to other routers in the network. Now, R0 reevaluates the channel already established for T2 and chooses the path R0-R2 instead of the earlier path through R1. In the case of LocDUCE, R0 is not aware of the channel established for T3 and continues to route traffic through the already established channel for T2. Thus, LocDUCE suffers degradation in performance for T2. This performance holds only for the condition that T3 has a higher utility than T2. In either case, it should be noted that the system-wide accrued utility will be lower than GloDUCE.

4.4 Comparison of LocDUCE and GloDUCE with Optimal Algorithm

In this section, we compare the performance of LocDUCE and GloDUCE with the optimal algorithm for small problem sizes. We used a simple setup to illustrate the performance of the proposed algorithms as it was computationally prohibitive to run the optimal algorithms on the setup used for the simulation experiments. The topology used for the simulations is shown in Fig. 14. We considered several scenarios for the comparisons. In one scenario, there were no downstream flows that affect the performance (we call this "Exp-I"). The simulation for measuring performance was conducted for homogeneous TUFs (step, soft_step and assoc individually) and heterogeneous TUFs (different TUFs in a single run). The thread TUFs are listed in Table 3 and Table 4. The results obtained for both LocDUCE and GloDUCE for Exp-I are shown in columns 2 and 3 of Table 6.

In the other set of experiments, the topology was slightly changed so that downstream traffic was generated by connecting two source nodes S4 and S5, one each to the routers R1 and R3, respectively. The experiments were repeated for two cases: 1) higher utility downstream traffic (Exp-II) and 2) lower utility downstream traffic (Exp-III). The results obtained are shown in the last four columns of Table 6 for Exp-II and Exp-III, respectively.

Table 6 shows that the performance of both LocDUCE and GloDUCE for Exp-I is close to the optimal. For the scenarios in Exp-II and Exp-III (where there are downstream loads), the performance of LocDUCE degrades, while the performance of GloDUCE is very close to the optimal.

TABLE 4
Thread TUF Characteristics

Threads	TUF	Max Utility
T1	Step	1.0
T2	Soft-Step	1.5
T3	Linear	2.0
T4	Quad	2.5
T5	Asoc	3.0

5 Implementation Experience

We implemented LocDUCE and GloDUCE in the Linux operating system (kernel version 2.4.18). The implementation includes: 1) an application layer UA routing module (URM) and 2) MAC layer UA packet scheduler (in the Linux kernel). The routing module implements both the LocDUCE and GloDUCE algorithms. The incoming application packets are captured off the interface and routed through the URM. Upon deciding the routes (based on the algorithm), the packets are placed on the corresponding outgoing links. The packets' normal path through the operating system is avoided by configuring a firewall and

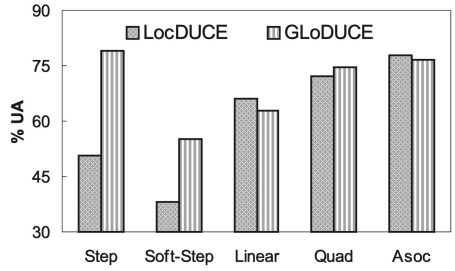


Fig. 11. Percentage UA comparison for different TUFs under LocDUCE and GLODUCE.

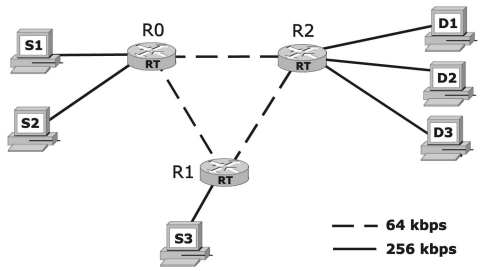
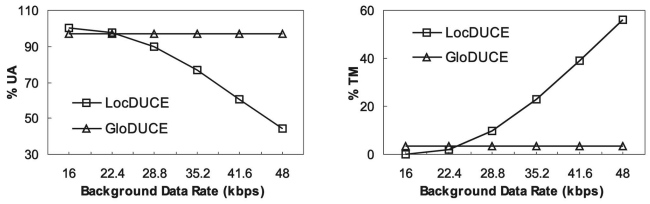


Fig. 12. Topology for downstream load simulation.



(a) (b)

Fig. 13. Performance comparison for T2 under increasing downstream load. (a) % UA. (b) % TM.

dropping duplicate packets. The complete implementation details can be found in [24].

5.1 Experimental Settings

Our experimental testbed is comprised of Linux machines configured as routers and interconnecting subnets, forming a WAN. The subnets contain source and destination nodes that host segments of distributable threads. The threads invoke operations on remote objects and generate real-time traffic.

Fig. 15 shows the network topology used in the study. The machines S1 and S2 are the source nodes and D is the destination. The machines R1, R2, R3, and R4 are the routers.

We considered six TUF shapes in our study. These include the step TUF, the plot correlation and track maintenance TUFs (referred to as "soft-step" TUFs hereafter), and the AWACS association TUF shown in Fig. 1a, Fig. 1b, and Fig. 1c, respectively. We also considered TUFs with linear, quadratic, and exponential shapes, as they are close variants of the AWACS TUF.

Our experimental scenarios include: 1) static and 2) dynamic. In the static scenario, message attributes such as data rate, message size, and interarrival time remain the same. In the dynamic scenario, we vary attributes including message laxity, arrival rate, message size, and utility variance. Table 7 shows TUFs of the threads and the source nodes generating the messages in the experiments. The maximum utility of each thread was 100 and messages' termination time is set to 4.0 seconds.

5.2 Static Scenario

Fig. 16 shows the % UA of OSPF, LocDUCE, and GloDUCE. The threads in this experiment were characterized by heterogeneous TUFs, i.e., each thread having a different TUFs (see Table 7). Percentage UA is the percentage of accrued aggregate utility to the maximum possible utility.

TABLE 5
TUFs of Different Threads

Threads	Time Constraint	Rate (kbps)
T1	Max Utility: 4.0 Termn. Time: 3.0 *sec.*	24
T2	Max Utility: 1.0 Termn. Time: 3.0 *sec.*	32
T3	Max Utility: 2.5 Termn. Time: 3.0 *sec.*	16–48

From Fig. 16, we observe that both LocDUCE and GloDUCE perform better than OSPF. The better performance of these algorithms over OSPF can be explained as follows: OSPF always selects the shortest path to a destination for any message. For example, for threads 2, 4, 5, and 6, OSPF takes the shortest path, R2-R3 to D. This decision of the algorithm does not optimize the message flows along all of the available paths. Although some versions of OSPF perform equal-cost load-balancing, the loads along the paths having the different costs are not balanced. Hence, when the shortest path becomes overloaded, the performance for all message flows degrades, irrespective of the TUFs.

Under LocDUCE or GloDUCE, the messages traverse diverse paths, one through R2-R3 and another through R2-R4-R3. Thus, the performance is better.

Fig. 17 shows the performance comparison between OSPF, LocDUCE, and GloDUCE in terms of % UA for some of the threads used in the static scenario, for exp, linear, quad, and asoc TUFs. Again, from Fig. 17, we observe that LocDUCE and GloDUCE perform better and accrue more utility than OSPF for all TUFs. This is because OSPF does not differentiate among the messages with different TUFs and, thus, does not prefer certain messages over others. LocDUCE does this differentiation and achieves higher system-wide and thread-level utility. Since GloDUCE performs global allocation, it prefers a lightly loaded route over a heavily loaded route and uses the entire path to make this decision. Thus, GloDUCE performs better than OSPF for all TUFs and, in some cases, outperforms LocDUCE (as shown for asoc TUF in Fig. 17).

5.3 Dynamic Scenario

For this scenario, we start four threads at node S2, which then invoke remote operations on node D. We increase the message arrival rate from two packets/sec to 10 packets/sec in steps of two.

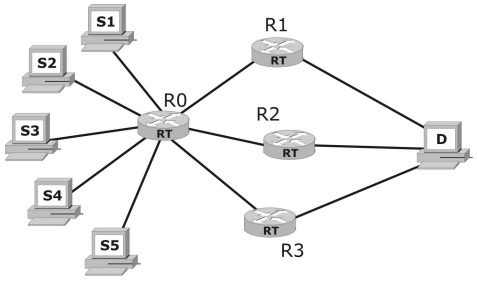


Fig. 14. Topology for comparison with the optimal algorithm.

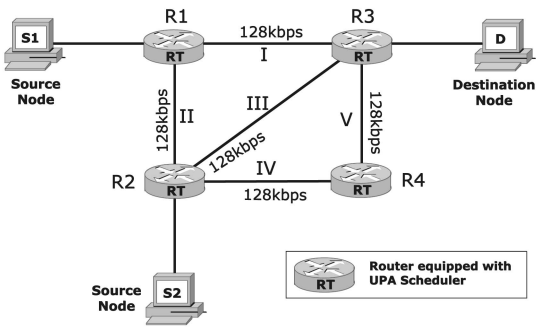


Fig. 15. Experimental network testbed.

Fig. 18 shows the % UA of OSPF, LocDUCE, and GloDUCE under increasing message arrival rates. Table 8 shows the average and standard deviation values for the experiments.

From Fig. 18, we observe that both LocDUCE and GloDUCE perform significantly better than OSPF, especially at higher arrival rates. Note that the data rate indicated is the individual rate for the messages. Thus, the actual load on the link is four times this value. Therefore, even under such heavily loaded conditions, we observe that LocDUCE and GloDUCE perform better. GloDUCE performs slightly better than LocDUCE.

The good performance of LocDUCE over OSPF is due to the fact that there are multiple paths to the destination, which are not explored by OSPF. We also observe that all message flows suffer the same performance degradation under OSPF. This is due to the fact that OSPF does not prefer one message flow over another and the scheduling is strictly first-in-first-out. From Table 8, we observe that the standard deviation for % UA is low. The same trend was observed for % TM. This indicates the consistency of the algorithm performance.

5.4 Downstream Loading

A study similar to the one performed in simulation (Section 4.3) was conducted in the implementation. In this study, the links between `R1` and `R3` in the testbed (Fig. 15) are considered to be downstream links and thread transitions from `S1` and `S2`, and terminate at `D`. The performance of LocDUCE and GloDUCE was compared and the results are plotted in Fig. 19.

From Fig. 19, we observe that the performance is identical to the simulation results. We can see that GloDUCE performs better than LocDUCE, accruing higher system-wide utility. We were unable to increase the background traffic beyond 112kbps. This is because of the limitations of the `libpcap` library and the `Berkeley packet filter` used in the prototype implementation.

Fig. 20 shows the implementation overheads for channel establishment and runtime.

TABLE 7
Experimental Parameters

Threads	TUF	Source Node
DT-1	`Step`	S1
DT-2	`Exp`	S2
DT-3	`Soft-step`	S1
DT-4	`Linear (Lin)`	S2
DT-5	`Quad`	S2
DT-6	`AWACS Assocn. (Asoc)`	S2

6 Past and Related Efforts

Real-time channels (or RTCs) were first defined in the work [16] of Ferrari and Verma. They proposed a scheme for establishing real-time channels in wide area networks. In this work, the authors established deterministic and statistical delay bounds for real-time traffic over wide area networks. In [18], Kandlur et al. develop a scheme for computing guarantees for delivery time of messages belonging to real-time channels. They use message scheduling and network flow control so that predictable delay bounds are obtained for packets. This scheme consists of two distinct phases: a channel establishment phase that selects the appropriate route; and a runtime message scheduling mechanism that ensures that the guarantees of the channel are not violated. They also propose a channel establishment algorithm.

Unlike [16] and [18], [19] proposes a parallel probe algorithm with different heuristics along different paths. They employ a *reservation phase*, in the forward direction from source to destination, to reserve resources and a *relaxation phase*, in the reverse direction, for releasing excess resources allocated during the previous phase. Admission control is performed during the reservation phase. If a call is rejected, resources reserved along the forward path are released.

A distributed route selection algorithm for RTCs is proposed in [20]. They employ the scheduling mechanism developed in [18]. Unlike in [18], this scheme accounts for flows currently being established. The algorithm searches for a route in parallel by flooding connection requests

TABLE 6
System-Wide Utility: Comparison of LocDUCE and GloDUCE with Optimal Algorithm (Normalized Values)

TUF Type	Without Downstream Traffic		With Downstream Traffic			
	Exp-I		Exp-II		Exp-III	
	LocDUCE	*GloDUCE*	*LocDUCE*	*GloDUCE*	*LocDUCE*	*GloDUCE*
Step	0.9995	0.9995	0.9510	1.0	0.9998	0.9998
Soft_Step	0.9992	0.9992	0.6281	0.9856	0.6996	0.9911
Trk_Asoc	0.9374	0.9374	0.8444	0.9136	0.9547	0.9547
Heterogeneous	0.8872	0.8872	0.7944	0.8521	0.8744	0.9633

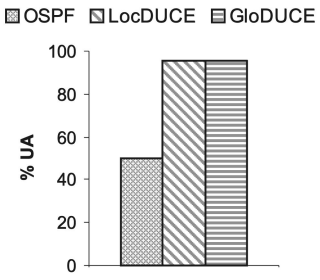


Fig. 16. Percent UA comparison.

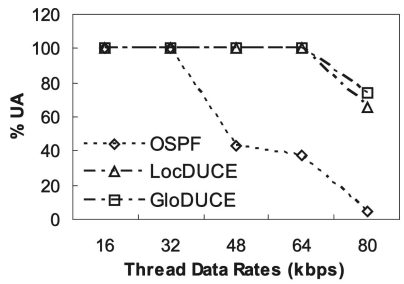


Fig. 18. Dynamic scenario performance under increasing rate of thread transitions.

through the network and prunes infeasible routes quickly. They use a multiclass earliest due date first packet scheduling algorithm at the interfaces. The worst-case delay for the message flows, in the path from source to destination, is computed using fixed priority scheduling.

The concept of dependable real-time communication is proposed in [25], where a dependable communication channel is one that consists of a primary channel and one or more backup channels. Sufficient resources are allocated along the primary channel. Fault tolerance is provided by allocating sparse resources along a backup channel. Raghavan et al. [26] propose a mixture of forward-recovery and detect, and detect and recover approach. This scheme uses bandwidth splitting and forward error correction. In addition to this, sparse resources are used for the detect and recovery approach outlined in [25]. Construction of backup channels, also called "segmented backups" is proposed in [27]. Unlike an end-to-end backup channel, a segmented backup consists of multiple backup channels spanning portions of a primary path.

In all the past approaches, the focus has been on deadline-based systems. To the best of our knowledge, there is no existing work that establishes real-time channels for messages based on TUFs. Most of the earlier schemes, like [20], propose some admission control schemes to limit link utilization from exceeding 100 percent. But, our approach does not perform an explicit admission control when a flow enters the system, but performs an implicit control when scheduling packets. Also, we consider overload scenarios for most cases. Shin et al. [20] also use a static priority-based algorithm for determining whether a particular flow satisfies the admission control and use a multiclass EDF algorithm for runtime packet scheduling.

Converting the multiclass EDF scheduler in [20] and replacing it with a TUF is not straightforward without being unfair to [20] or our approach. One example method to adapt the multiclass EDF scheduler to our model is: 1) ignoring the non-real-time queue, 2) expressing deadlines using `step` TUFs having the same nonzero utility value until deadline time and zero after that, and 3) disabling admission control and admitting all the flows for considering overload scenarios. Allowing flows that would otherwise have been rejected by admission control would result in extremely poor performance for the approach in [20]. This is because EDF is known to suffer from the "domino effect" problem for utilization values beyond 100 percent [12]. Also, the message models employed in [20] and our approach are different and the delay analysis derived for both approaches do not match.

7 Conclusion and Future Work

In this paper, we present utility accrual channel establishment algorithms, LocDUCE and GloDUCE, for multihop networks. LocDUCE performs localized UA-based channel allocation without considering other real-time channels established in the network. On the other hand, GloDUCE performs global allocation of channels, and considers channels established through other routers in the network. Our simulation studies and implementation measurements reveal the effectiveness of the algorithms and their superior performance with respect to the OSPF protocol. We also observe that the algorithms perform close to the optimal algorithm for some cases.

LocDUCE maximizes total utility, by performing local optimization. This makes the algorithm relatively simple (with respect to GloDUCE). The volume of state information stored at a router is limited to the channels established through that router. Since LocDUCE performs a local decision, there are situations when the algorithm's

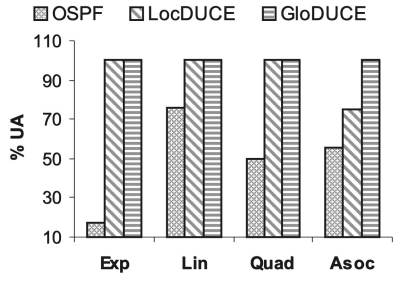


Fig. 17. Thread performance under a static scenario.

TABLE 8
Average and Standard Deviation

Rates (kbps)	% UA					
	OSPF		LocDUCE		GloDUCE	
	Avg.	*Dev.*	*Avg.*	*Dev.*	*Avg.*	*Dev.*
16	99.96	0.0024	99.94	0.002	99.97	0.002
32	99.94	0.0314	99.94	0.001	99.95	0.001
48	43.33	0.0600	99.96	0.001	99.94	0.002
64	37.51	0.0978	99.95	0.005	99.93	0.001
80	5.050	0.1248	66.05	0.50	73.84	0.495

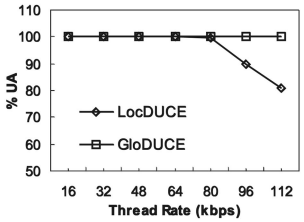


Fig. 19. Percent UA comparison under increasing downstream loads.

performance degrades. For example, when the downstream traffic has a higher utility than upstream traffic, LocDUCE performs worse than OSPF in some cases. On the contrary, GloDUCE performs a global optimization and accrues much higher system-wide utility than LocDUCE. In most situations, both algorithms perform identically. However, when downstream traffic has higher utility than upstream traffic, GloDUCE performs better. Thus, we envision that this scheme is best suited for mesh networks, with shorter path lengths from source to destination.

For GloDUCE, each intermediate router maintains information on all existing channels in the network. When a new channel is created, this information is broadcast to all routers in the system. This can possibly flood the network with channel update packets, especially when application packet arrivals are skewed.

Some aspects of the work are directions for further research. Examples include establishing channels while providing stronger assurances on message timeliness behavior (individual and collective) and tolerating link and node failures.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their helpful comments and suggestions. This work was partially supported by the US Office of Naval Research under Grant N00014-00-1-0549. E.D. Jensen's participation was sponsored by the MITRE Technology Program.

REFERENCES

[1] OMG, "Real-Time CORBA 2.0: Dynamic Scheduling Specification," technical report, Object Management Group, Sept. 2001, OMG Final Adopted Specification, http://www.omg.org/docs/ptc/01-08-34.pdf.

[2] E.D. Jensen, A. Wellings, R. Clark, and D. Wells, "The Distributed Real-Time Specification for Java: A Status Report," *Proc. Embedded Systems Conf.,* 2002.

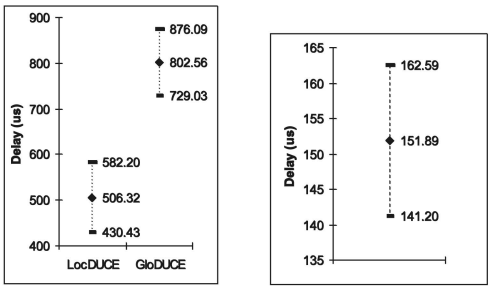
[3] J.D. Northcutt, *Mechanisms for Reliable Distributed Real-Time Operating Systems—The Alpha Kernel.* Academic Press, 1987.

[4] The Open Group Research Inst.'s Real-Time Group, *MK7.3a Release Notes.* Cambridge, Mass.: The Open Group Research Inst., Oct. 1998.

[5] GlobalSecurity.org, "Multi-Platform Radar Technology Insertion Program," http://www.globalsecurity.org/intell/systems/mp-rtip.htm/, Apr. 2005.

[6] GlobalSecurity.org, "BMC3I Battle Management, Command, Control, Communications and Intelligence," http://www.globalsecurity.org/space/systems/bmc3i.htm/, Apr. 2005.

[7] W. Horn, "Some Simple Scheduling Algorithms," *Naval Research Logistics Quarterly,* vol. 21, pp. 177-185, 1974.



(a) (b)

Fig. 20. Implementation overhead. (a) Establishment. (b) Runtime.

[8] E.D. Jensen, C.D. Locke, and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS),* pp. 112-122, 1985.

[9] R. Clark et al., "An Adaptive, Distributed Airborne Tracking System," *Proc. Workshop Parallel and Distributed Real-Time Systems (WPDRTS),* pp. 353-362, Apr. 1999.

[10] D. Maynard et al., "An Example Real-Time Command, Control, and Battle Management Application for Alpha," technical report, Computer Science Dept., Carnegie Mellon Univ., Dec. 1988.

[11] P. Li, "Utility Accrual Real-Time Scheduling: Models and Algorithms," PhD dissertation, Virginia Tech, 2004.

[12] C.D. Locke, "Best-Effort Decision Making for Real-Time Scheduling," PhD dissertation, Carnegie Mellon Univ., 1986.

[13] R. Clark, "Scheduling Dependent Real-Time Activities," PhD dissertation, Carnegie Mellon Univ., 1990.

[14] J. Wang and B. Ravindran, "Time-Utility Function-Driven Switched Ethernet: Packet Scheduling Algorithm, Implementation, and Feasibility Analysis," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 1, Jan. 2004.

[15] H. Wu, B. Ravindran, E.D. Jensen, and U. Balli, "Utility Accrual Scheduling under Arbitrary Time/Utility Functions and Multiunit Resource Constraints," *Proc. IEEE Int'l Conf. Embedded Systems and Real-Time Computing Systems and Applications (RTCSA),* pp. 80-98, Aug. 2004.

[16] D. Ferrari and D.C. Verma, "A Scheme for Real-Time Channel Establishment in Wide Area Networks," *IEEE J. Selected Areas in Comm.,* vol. 8, no. 3, pp. 368-379, Apr. 1990.

[17] V. Kompella, "Multicast Routing Algorithms for Multimedia Traffic," PhD dissertation, Univ. of California, San Diego, 1993.

[18] D.D. Kandlur, K.G. Shin, and D. Ferrari, "Real-Time Communication in Multi-Hop Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 10, pp. 1044-1056, Oct. 1994.

[19] G. Manimaran, H.S. Rahul, and C.S. R. Murthy, "A New Distributed Route Selection Approach for Channel Establishment in Real-Time Networks," *IEEE/ACM Trans. Networking,* vol. 7, no. 5, pp. 698-709, Oct. 1999.

[20] K.G. Shin, C. Chou, and S. Kweon, "Distributed Route Selection for Establishing Real-Time Channels," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 3, pp. 318-335, Mar. 2000.

[21] G.L. Lann, "Proof-Based System Engineering and Embedded Systems," *Lecture Notes on Computer Science,* G. Rozenberg and F. Vaandrager, eds., vol. 1494, pp. 208-248. Springer-Verlag, Oct. 1998.

[22] K. Channakeshava and B. Ravindran, "On Utility Accrual Channel Establishment in Multi-Hop Networks," *Proc. IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing (ISORC),* pp. 277-284, May 2004.

[23] K. Chen and P. Muhlethaler, "A Scheduling Algorithm for Tasks Described by Time Value Function," *J. Real-Time Systems,* vol. 10, no. 3, pp. 293-312, May 1996.

[24] K. Channakeshava, "Utility Accrual Real-Time Channel Establishment in Multi-Hop Networks," master's thesis, Virginia Tech, Aug. 2003, http://scholar.lib.vt.edu/theses/available/etd-0322004-173216/.

[25] S. Han and K.G. Shin, "A Primary-Backup Channel Approach to Dependable Real-Time Communication in Multi-Hop Networks," *IEEE Trans. Computers,* vol. 47, no. 1, pp. 46-61, Jan. 1998.

[26] S. Raghavan, G. Manimaran, and C.S.R. Murthy, "An Integrated Scheme for Establishing Dependable Real-Time Channels in Multi-Hop Networks," *Proc. Eighth Int'l Conf. Computer Comm. and Networks (ICCCN)*, Oct. 1999.
[27] K.P. Gummadi, M.J. Pradeep, and C.S. R. Murthy, "An Efficient Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, pp. 81-94, Feb. 2003.

Karthik Channakeshava is a PhD student in the Bradley Department of Electrical and Computer Engineering at Virginia Tech. He received the master's degree in computer enginering from the same university. He received the BEng degree in electrical engineering from Bangalore University. He is currently working on energy management issues for resource constrained nodes in ad hoc and sensor networks. His research interests include distributed real-time systems, ad hoc and sensor networks, and network quality of service. He is a student member of the IEEE.

Binoy Ravindran is an associate professor in the Bradley Department of Electrical and Computer Engineering at Virginia Tech. He is fascinated and challenged by building adaptive real-time software, i.e., real-time application and system software that can dynamically adapt to uncertainties in their operating environments and satisfy time constraints, acceptably well with acceptable predictability, according to application-specific criteria. Toward that end, he focuses on time/utility function (TUF)/utility accrual (UA) real-time scheduling and resource management—an adaptive time-critical resource management paradigm invented by Doug Jensen (almost 35 years ago) and a central concept behind the Alpha distributed real-time kernel. His students have recently developed several new results on TUF/UA scheduling and resource management, including those on stochastic scheduling, distributed scheduling, energy consumption, memory management and garbage collection, and nonblocking synchronization. Many of these new results have been transitioned for use in US Department of Defense programs. He is a senior member of the IEEE and a member of the IEEE Computer Society.

E. Douglas Jensen is the consulting scientist of the Information Technologies Directorate at the MITRE Corporation. His principal focus is on time-critical resource management in dynamic distributed object systems, particularly for combat platform and battle management applications. He directs and conducts research, performs technology transition, and consults on US Department of Defense programs. He joined MITRE from previous positions at HP, Digital Equipment, and other computer companies. From 1979 to 1987, he was on the faculty of the Computer Science Department at Carnegie Mellon University, where he created and directed the largest academic real-time research group of its time. Prior to that, he was employed in the real-time computer industry, where he engaged in research and advanced technology development of distributed real-time computer systems, hardware, and software. He is considered one of the original pioneers, and leading visionaries, of distributed real-time computer systems and is widely sought throughout the world as a speaker and consultant. He is a member of the IEEE.