End-to-End Delay Bound for Prioritized Data Flows in Disruption-tolerant Networks

Md Yusuf Sarwar Uddin, Fatemeh Saremi, Tarek Abdelzaher
Department of Computer Science
University of Illinois at Urbana-Champaign
{mduddin2, saremi1, zaher}@illinois.edu

Abstract

This paper computes end-to-end delay bounds for prioritized data flows in disruption-tolerant networks (DTNs). DTNs suffer intermittent connectivity among nodes due to node mobility. When deployed in mission-critical applications, such as disaster response, an interesting question becomes to quantify end-to-end packet delays under assumptions on node mobility. In this paper, we answer this question for the special case of DTNs with recurrent mobility patterns. A recurrent pattern refers to one where nodes revisit the same locations repeatedly. We devise a suitable model for recurrent DTNs that captures their timing and mobility properties. We then apply the recently proposed delay composition algebra to the resulting network model in order to determine an upper bound on end-to-end communication delays of network flows. Evaluation results show that the upper bound is moderate in its pessimism and can be used for deployment planning purposes.

1 Introduction

This paper develops an analysis technique for quantifying end-to-end delays of packets in disruption-tolerant networks. Disruption-tolerant networks [4] are those where nodes suffer intermittent connectivity due to mobility. No connected path is likely to exist between a source and a destination at any given time. Consider, for example, the scenario where some natural disaster, such as a hurricane, tsunami, or severe earthquake, disrupts power lines, wired communication infrastructure, and cell-phone towers. Without power and communication, relief efforts may need to rely on battery operated wireless devices, possibly installed in relief centers, put in rescue vehicles, and carried by volunteer workers, to propagate information. Many individuals and vehicles will perform repetitive tasks (e.g., move supplies between locations, or patrol troubled areas). Collectively they will form an instance of a disruption-tolerant network, where information can propagate through series of recurrent node encounters. A question becomes, how long does it take to transfer information across this network from specific sources to destinations, given some knowledge of recurrent node mobility patterns? The question is primarily hypothetical and used for planning only. During actual execution, it is impossible to measure global network state. Nevertheless, it is important to understand the relation between mobility patterns, load, and resulting delay, if one is to guess what network performance might be under given deployment conditions. This paper presents analytic results that constitute a first step towards achieving such an understanding. The results of this paper can be used to quantify network performance under assumptions on load and mobility.

Unlike previous work on distributed real-time systems, where resources were assumed to be connected, in our model, messages are transferred through the network along a sequence of short-lived node contacts (i.e., encounters between nodes). Two nodes are said to be in contact when they are close enough to establish a connection and transfer packets. Usually, there is a long time gap between such contacts, compared to the duration of a contact. For example, moving vehicles might occasionally pass landmarks where wireless devices with some data storage capacity were planted to act as “mailboxes”. When packets are forwarded to a node, these packets reside on the node and are physically carried around until the node encounters another, possibly forwarding the packet onwards. The sum of all residence delays of a packet at different nodes (until delivery) constitutes its end-to-end delivery delay.

In this context, our paper addresses the following problem. Given a set of prioritized data flows of arbitrary source-destination pairs in a disruption-tolerant network, we compute an end-to-end packet delivery delay bound for each flow. One of the challenges of computing packet delivery delay is to understand the temporal and spatial behavior of the moving nodes and to represent the network in some
analyzable form that allows us to derive delay expressions. In our earlier work [29], we introduced a disruption-tolerant network model where the network is modeled by an inter-contact graph. This graph abstracts the physical network into a collection of contacts and the associated time gaps between those contacts. In this paper, we revisit the same graph with further annotations to allow modeling end-to-end delay. To the best of our knowledge, this is the first work in the real-time systems literature that attempts to provide delay bounds for disruption-tolerant networks.

The paper uses delay composition algebra to analyze inter-contact graphs. Delay composition algebra allows reducing distributed systems to centralized systems that are equivalent to the original systems in terms of timing properties. Hence, we show how to convert disruption-tolerant network models into a form analyzable by delay composition algebra, so that it can be used to compute delay bounds. As might be expected, the resulting delay bounds are not tight. Evaluation shows, however, that the pessimism is moderate. The approach does provide a meaningful estimate of worst-case network behavior given recurrent mobility patterns.

The rest of this paper is organized as follows. Section 2 describes existing real-time literature on delay estimation techniques and issues of delay computation pertaining to disruption-tolerant networks. Section 3 describes our proposed network model for DTNs and the data flow model that we consider for delay analysis. Section 4 presents the techniques of computing end-to-end delay bounds of data flows. Section 5 presents simulation results, followed by conclusion (Section 6).

2 Related Work

Estimation of delivery delay in disruption-tolerant networks has been addressed in previous work. In most cases, the delay is used as a routing protocol metric for selecting routes. The repository at [4] contains some recent work on this problem and other DTN-related design issues.

Early work [7] defines the DTN routing problem and presents different aspects of the problem depending on available knowledge at nodes. The authors model the DTN as a time-dependent graph, edges of which are annotated with a delay attribute. A minimum delay path is then computed from this graph through running a modified version of Dijkstra algorithm. In [19], the authors model the network with a space-time graph consisting of different layers, each of which represents a copy of the DTN in one time slot of the network’s activity and contains encounters in that time slot. A shortest path algorithm is utilized to calculate the path with minimum delay.

Another approach [21] considers deterministic mobility and solves the routing problem subject to network constraints utilizing algorithms based on the model introduced in [7]. Several papers [16, 17, 18] concentrate on a non-deterministic DTN with cyclic mobility patterns and model the network as a space-time graph. They construct a probabilistic state-space graph and obtain the optimal routing scheme in terms of expected delivery latency as a solution of a Markov decision process. Others [6] consider a deterministic and centralized DTN and construct a time-independent graph wherein vertices denote replicated nodes at contact times and edges denote corresponding contacts. All contacts are hence enumerated by edges and indexed by time in a very large graph. Then they exploit graph algorithms to achieve optimal results subject to network constraints.

RAPID [1] assumes an exponential distribution for contact delays between a pair of nodes and computes the expected delivery delay by taking a minimum over all such delays for multiple copies of the same packet. The computed delays are translated into utility values that are used when nodes make forwarding decisions. The authors in [26] explore the single-copy routing space and propose a framework to derive upper and lower bounds on delay. The authors use this framework to analyze the performance of multi-copy routing scheme as well [25].

Inter-contact routing [29] exploits recurrence of meeting patterns, identifies “reliable” sequences of recurrent contacts as an end-to-end path to destinations, and computes path delay as the sum of inter-contact delays along a path. Inter-contact routing introduces a convenient graph representation for DTNs, modeling the network as a collection of recurrent contacts and time gaps between pairs of contacts. Inter-contact routing does not assume a known distribution for contact delays. Instead, it computes mean and variance of delay among contacts and stores the values as measures of delay distribution.

Although the efforts described above compute delivery delay as a routing metric, none addresses the problem of computing end-to-end delay bounds for prioritized flows. When more than one class of traffic exists, as might be agreed upon per disaster recovery protocols, it is important to understand the impact of prioritization on delay bounds.

In real-time literature, there exists a large volume of work that computes timing properties of distributed systems. Accurate analysis methods, such as [32, 5], construct a precise schedule of length equal to the hyper-period. Offline schedulability tests [14, 33, 20] analyze distributed systems by dividing end-to-end deadlines of tasks into per-stage deadlines. Holistic schedulability analysis [28, 23, 24] considers the worst-case delay of each stage of computation to derive the jitter of next stage, adding up delay across paths. Network Calculus [2, 3, 27, 12, 31] analyzes the network one node at a time. They model arrival patterns of flows at a particular node and the node’s scheduling policy
using arrival and service curves, respectively. Based on this information, the rate of departing flow which would serve as the arrival curve for the next node can be determined. Comparisons of holistic analysis and network calculus are presented in [15]. Delay Composition Algebra was recently introduced [8, 9, 10, 11] as a reduction-based approach, which reduces the entire distributed system to a hypothetical single node. It takes into account execution pipelining effects between subtasks running in a distributed system and provides a good upper bound for end-to-end delay of tasks. None of these approaches, however, is designed for analysis of disconnected networks. They all assume that the distributed system is connected.

3 Modeling DTNs

Delays in real-time systems arise when tasks wait for resources to become available (e.g., to finish executing higher-priority tasks). Generally, resources may be processors, communication links, or anything that can be used by one task at a time. In a mobile environment (e.g., DTNs), we need to consider availability in space as well as in time. In this environment, a resource (say, a communication link) may be unavailable because it is busy (allocated to someone else at this time) or out of range (not available at the current location). Hence, one must quantify wait times as a function of not only workload, but also mobility patterns of nodes.

Rather than modeling the complex spatial behavior of nodes, we model the effects of such spatial behavior on the topological properties of interest (namely, connectivity and wait times). In particular, we are interested in identifying properties that can affect the end-to-end delay of packet delivery. Observe that complex spatial behavior models are not always necessary for this purpose. For example, in the architecture community, rather than understanding the exact “mobility patterns” of the program counter (in the space of memory locations), it was possible to achieve significant performance benefits simply by exploiting an over-arching behavior principle such as locality of reference, which leads to caching. Similarly, in DTNs, to route packets, we exploit an overarching node behavior principle; namely, recurrency. We consider a category of networks where nodes perform recurrent jobs such as moving supplies between given locations, patrolling given neighborhoods, and shuttling between hospitals and relief centers. The network is thus formed of fixed nodes at certain key locations and mobile nodes that visit them often in a given order recurrently. Node contacts represent the main means for forwarding packets. We directly capture the delay properties of recurrent contacts, thus resulting in a very simple model for DTNs, as described below.

3.1 Network Model

We consider a set of static and mobile nodes extended over a geographic area, where they remain disconnected, except when contacts occur. At each contact, packets from one node can be transferred to the other. Packets, originating at source nodes, can be forwarded via a sequence of contacts until they reach their destinations. Hence, instead of using the traditional network model consisting of network nodes and links, a DTN can best be described by contacts (encounters of pairs of nodes) and the timing relations between them.

We denote contacts by the corresponding node names. For example, contact \( ij \) denotes an encounter between node \( i \) and node \( j \). Much like with periodic task models, if a pair of nodes meet repeatedly at a fixed schedule, a time series can be used to enumerate all future meeting times. (Later, we relax the need to know future meeting times.) Let \( T_{ij}^{(t)} \) denote the \( t \)-th meeting time between node \( i \) and \( j \). The meeting time indicates the starting time of the corresponding contact, although the contact may continue for some duration. The time series for each contact enlists all meetings for a given pair of nodes. We represent the network, \( N \), as a collection of time series of all such contacts:

\[
N = \left\{ T_{ij}^{(l)} \mid 0 \leq l < \infty, \forall i,j, \text{with } i \neq j \right\} \tag{1}
\]

We assume that for all contacts \( ij \), \( T_{ij}^{(0)} = 0 \). If two nodes never meet, we set, \( T_{ij}^{(1)} = \infty \). We define a few terms as follows:

\[
\delta^{(l)}(ij) = T_{ij}^{(l+1)} - T_{ij}^{(l)} \tag{2}
\]

\[
\delta^{(l)}(ij \rightarrow jk) = T_{jk}^{(l)} - T_{ij}^{(l)} \tag{3}
\]

\[
l' = \arg \min_h \left\{ T_{jk}^{(h)} \mid T_{jk}^{(h)} > T_{ij}^{(l)} \right\} \tag{4}
\]

where \( \delta^{(l)}(ij) \), called contact interval, denotes the time gap between \( l \) and \( (l + 1) \)-th \( ij \) encounters and \( \delta^{(l)}(ij \rightarrow jk) \), called inter-contact interval, denotes the time gap between the next immediate encounter of \( jk \) followed by \( l \)-th \( ij \) encounter. Semantically, \( \delta^{(l)}(ij) \) denotes the waiting time of node \( i \) for the next immediate transfer opportunity to node \( j \) after the \( l \)-th encounter, and \( \delta^{(l)}(ij \rightarrow jk) \) denotes the waiting time of a packet, at node \( j \), for the next immediate transfer opportunity to node \( k \) after node \( j \) has received a packet from node \( i \) at their \( l \)-th encounter.

In addition to meeting times, each contact is associated with another quantity, called contact duration. Contact duration measures the time span of a contact during which two nodes can remain in active radio communication by exchanging packets. We denote \( \gamma^{(l)}(ij) \) as the contact dura-
tion of $l$-th $ij$ contact. The network, $N$, can now be described as a time series graph, $G = (C, \Gamma, E)$, where $C$, the set of contact intervals, $\Gamma$, the set of contact durations, and $E$, the set of inter-contact intervals, are given by the following set of time series:

$$
\mathcal{C} = \left\{ \delta_{ij}^{(l)} \right\}_{0 \leq l < \infty, \forall i,j, \text{with } i \neq j}
$$

$$
\Gamma = \left\{ \gamma_{ij}^{(l)} \right\}_{0 \leq l < \infty, \forall i,j, \text{with } i \neq j}
$$

$$
E = \left\{ \delta^{(l)}(ij \rightarrow jk) \right\}_{0 \leq l < \infty, \forall i,j,k, \text{with } i \neq j \neq k}
$$

The end-to-end packet delivery delay between a pair of nodes can be expressed as a summation of contact intervals and inter-contact intervals. Suppose, a packet $P$ originates at node $i$ and takes the contact sequence $ij, jk$, to reach node $k$, i.e., node $i$ forwards the packet to $j$ (at contact $ij$) and then $j$ forwards to $k$ (at contact $jk$). For brevity, let the packet be created at $T^{(ij)}(ij)$ (if not, a certain offset can be added). Assuming the actual packet transmission time on an active connection is zero, the end-to-end delay is simply the wait time between two contacts. Hence, $\text{delay}(P) = \delta^{(l)}(ij \rightarrow jk)$ (Figure 1a).

\begin{figure}[h]
\centering
\includegraphics[width=\linewidth]{delay_computation.png}
\caption{An example of delay computation}
\end{figure}

This delay is however exact only if the packet can be successfully transferred at the very first occurrence of each contact. Depending on the number of other packets waiting for the same contact, the packet may not be able to be transferred to the next peer on the first contact; it may be delayed until the next contact or beyond. Figure 1b illustrates the case where $P$ is transferred to node $j$ on third $ij$ contact, and to node $k$ on second $jk$ contact. Hence, the end-to-end delay is given by:

$$
\text{delay}(P) = \sum_{l=0}^{t+1} \delta^{(l)}(ij) + \delta^{(l+2)}(ij \rightarrow jk) + \delta^{(l')}([jk])
$$

One obvious problem with the above expression is that it requires knowledge of the entire future time series of contacts. The complete time series of contacts for a large network operating for a moderately long time would be very hard and expensive to collect. One possibility is to replace the exact time series with the probability distributions of contact intervals. Given delay distributions for all contacts and edges in the graph, one can probabilistically compute delays between contacts. Since we are interested in the worst case delay bound, however, we instead consider the worst case time separation between two successive contacts. We define the following three terms:

$$
\delta(ij) = \sup \{\delta^{(l)}(ij)\}, \forall i,j
$$

$$
\gamma(ij) = \inf \{\gamma^{(l)}(ij)\}, \forall i,j
$$

$$
\delta(ij \rightarrow jk) = \sup \{\delta^{(l)}(ij \rightarrow jk)\}, \forall i,j,k
$$

In practice, the above quantities may be estimates of the maximum delays between pairs of successive contacts. For contact duration, it denotes the minimum span. If we replace all $\delta^{(l)}$'s for a certain contact, say $ij$, by a single $\delta(ij)$ value, we obtain a purely periodic network, in that $\delta(ij)$ denotes the contact period of the contact $ij$. Semantically, $\delta(ij)$ denotes the longest time gap by which node $i$ or $j$ expects to encounter the next contact $ij$, no matter when they start waiting for the contact. Similarly, $\delta(ij \rightarrow jk)$ denotes the longest possible time that node $j$ waits to encounter node $k$ after it meets node $i$. Obviously, while $\delta(ij)$ and $\delta(ji)$ are the same, $\delta(ij \rightarrow jk)$ and the reverse $\delta(kj \rightarrow ji)$ are not. To appreciate the difference, consider a patrol that circles a given neighborhood on a specified path. The length of one round may be an hour, but the time between passing intersection $i$ and the subsequent intersection $k$, by patrol $j$, may by only a few minutes. Once $k$ is passed, it takes the rest of the hour to reach $i$ again. Hence, $\delta(ij \rightarrow jk)$ is much shorter than $\delta(kj \rightarrow ji)$.

Based on the above discussion, we define our DTN model as follows. The network is described by a time-invariant directed graph, called inter-contact graph, $G = (C, E)$, where $C$ is the set of encounters and $E$ is the set of edges between encounters. Each vertex $c$ is annotated by a tuple $(\delta(c), \gamma(c))$, which measures the contact period and contact duration of $c$, respectively. To avoid ambiguity, we use the term “vertex” to denote a contact in the inter-contact graph, whereas “node” means the physical device. We use $c_1 \rightarrow c_2$ to denote an edge that expresses the occurrence of contact $c_2$ followed by contact $c_1$. Two contacts have an edge if they share a common node between them. We write the shared node in the inner side of the edge expression for better understanding of the sequence of packet transfers. For example, $ij \rightarrow jk$ implies the transfer of packets in a sequence $i \rightarrow j \rightarrow k$. The reverse edge between the same contacts is written as $kj \rightarrow ji$. Once we adopt the above notations, we can compute the end-to-end delay of a packet in terms of $\delta$'s. For example, Equation 8 now becomes, $\text{delay}'(P) = \delta(ij \rightarrow jk) + 2 \times \delta(ij) + \delta(jk)$. Obviously, $\text{delay}'(P) \geq \text{delay}(P)$.
For each contact $c$, we introduce another term \textit{transfer volume}, $v(c)$, which indicates the minimum number of bytes that a node can transfer onto the peer node upon contact $c$. The transfer volume of a contact is determined by the contact duration and the transmission bandwidth of the link associated between the node pair. For a transmission rate of $R$ (byte/sec), the transfer volume is given by $v(c) = R \times \gamma(c)$. Assuming the symmetry of the channel, we consider $v(ij) = v(ji)$.

We denote a path or route, in the inter-contact graph, as a sequence of vertices (i.e., contacts) $\{c_1, c_2, \ldots, c_k\}$ such that each successive edge $c_i \rightarrow c_{i+1}$ exists in the inter-contact graph. The first and last contact contain the source and destination node respectively. A path explicitly enumerates the sequence of contacts, and in turn nodes, that a packet follows when forwarded along the same path. The path can also be written as $c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_k$.

The inter-contact graph, $G$, expresses the topographical properties of the network. Any timing property of the network, particularly the end-to-end delay, can be computed in terms of $\delta$’s and $v$’s of $G$. All such delays are, however, an over-estimation of the actual delays computed from $G$. Since we are interested in worst case delay bound, such over-estimation is allowed.

### 3.2 Data Flow Model

We define a \textit{data flow} to be a set of packets emanating from a source node that traverse a given path to reach a certain destination. Multiple data flows may be forwarded at the same contact or a set of contacts. We consider prioritized flows in that each flow is assigned an explicit priority value. Without loss of generality, we assume that flow ID and its priority are the same and lower ID indicates higher priority, flow 1 being the highest priority flow. Each flow $k$ is assigned a path, $path_k$, that enumerates the sequence of contacts along which packets from the same flow are forwarded. Later, we use the notion of flow and path interchangeably. All packets in a flow $k$ have the same size, $p_k$. Packets can be injected periodically or sporadically. For periodic flows, $P_k$ denotes the period of flow $k$, which is the time interval between two successive packets injected by the source. For aperiodic flows, $P_k$ denotes the minimum time gap between two successive packets.

Although the inter-contact graph may have many vertices and edges, in the following, we shall only consider the subgraph that contains flows. Edges and vertices that are not part of any flow are removed. This subgraph is called the \textit{data flow graph}.

We need to explain why data flows are denoted as a sequence of contacts, not nodes. It is obvious that packets will be transferred from one node to another. Recall that our objective is to compute the end-to-end delay for a set of data flows. So, we need to identify the entities where competition among flows happens, in that a flow (possibly a higher priority one) imposes delay on other flows (of lower priority). These points of competition are the contacts. Only flows that need to be forwarded at the same contact compete. We assume that memory is sufficient. Hence, packets that reside on a node, waiting for different contacts, are not in competition.

### 4 End-to-end Delay Bound for DTNs

In this section, we describe how an end-to-end delay bound is computed for fixed-route prioritized data flows in DTNs. We first analyze the various components of the delay and then use delay composition algebra to compute the delay bound.

#### 4.1 Delay Components

As modeled by our data flow model, each packet from a particular flow follows a certain sequence of contacts. The time gap between two successive contacts on the path is given by the inter-contact delay. Once a contact occurs, a connection (i.e., link) between the two corresponding nodes is established and packets are transferred from one node to another in priority order. Due to the limited transfer volume, it may not be possible to transfer all packets in the buffer to the next node. Some packets may need to wait for a future occurrence of the contact. For a particular packet, we define \textit{transfer delay} to be the time between the instant of first encounter with the next-hop node and the instant when it is successfully transferred. Note that, the inter-contact delay is the time between different contacts, whereas the transfer delay is caused by waiting for returns of the same contact. The sum of inter-contact delays and transfer delays for all contacts along the path constitutes the total end-to-end delay of a particular flow.

We assume for now that actual transmissions of packets over an active connection happen in zero time. This assumption is based on the fact that physical packet transmission time along a link is negligible compared to the time a node waits for contact with another in a DTN. We relax this assumption later on.

In Figure 2a, we demonstrate the delay trajectory of a packet from flow $c_1 \rightarrow c_2 \rightarrow c_3$. We observe from the figure that the packet required 2 occurrence of $c_1$ contact, 1 $c_2$ contact and 3 $c_3$ contacts to make a successful transfer onward and for the final delivery. There are also two inter-contact delays for two edges $c_1 \rightarrow c_2$ and $c_2 \rightarrow c_3$. There is also a certain amount of waiting time involved prior to the very first contact $c_1$ after the packet was originated. This delay is smaller than $\delta(c_1)$, the contact period of $c_1$. We safely assume this waiting delay to be $\delta(c_1)$. In subsequent
The end-to-end delay for a certain flow is determined as the summation of all inter-contact delays and sum of zero or some integer multiple of contact periods along the path. The delay is given by the following general expression, for flow $k$:

$$\text{delay}(k) = \sum_{c_j} \delta(c_j \rightarrow c_{j+1}) + \sum_{c_j} x_j \delta(c_j)$$  \hspace{1cm} (12)

where $c_j$'s are the contacts in $\text{path}_k$ and $x_j (\geq 0)$ denotes the number of times contact $c_j$ has been missed by the flow. We denote $x_j \delta(c_j)$ as the “transfer delay” at contact $c_j$ that measures the amount of time the sender node waits on the same contact until it becomes able to transfer the packet onto the next contact. While the sum of inter-contact delays for a flow is entirely given by the path of the flow, which does not depend on the presence of other flows, the transfer delay is greatly influenced by the presence of other flows, particularly by higher priority flows that share one or more contacts with flow $k$. One subtle issue is to compute $x_j$, counting the number of contact misses that a certain flow suffers due to the presence of other higher priority flows. Given the data flow graph, the $x_j$'s are the only unknowns in computing the end-to-end delay bound for a flow. Estimating the worst-case values of $x_j$'s is the purpose of our analysis.

To ease the analysis, we order the terms in the delay summation putting the sum of all inter-contact delays first ahead of the transfer delays as suggested by Equation (12) and depicted in Figure 2b. This split does not affect the total end-to-end delay. Let the sum of inter-contact delays, end-to-end contact delay, be $D_c(k)$ and the sum of transfer delays and the last contact duration, end-to-end transfer delay, be $D_t(k)$ for flow $k$. The total end-to-end delay, denoted by $\text{delay}(k)$, is therefore given by, $\text{delay}(k) = D_c(k) + D_t(k)$.

Splitting the transfer delay from the contact delay separates path-specific delays (the wait for first contact with next hop) from load-specific delay (the wait for a number of returns of that contact before a packet’s turn comes to be forwarded). The latter delay is a step function. As described in the next section, this step function can be upper-bounded by a straight line. The slope of the straight line can be interpreted as link bandwidth in a virtual connected network. Hence, the original path delay of flow $k$ can be decomposed into a leading delay $D_c(k) = \sum_{c_j \in \text{path}_k} \delta(c_j \rightarrow c_{j+1})$ plus the path delay, $D_t(k)$, through a virtual connected network with point-to-point links, that upper bounds $D_t(k) = \sum_{c_j \in \text{path}_k} x_j \delta(c_j)$. Real-time literature has recent results to compute delay (upper) bounds on $D_t(k)$. In particular, we use delay composition algebra [9] for that purpose.

Although we assume that the packet transmission times are negligible and are not added to the delay bound, they are indeed considered. The packet transmission delays are actually overlapped with the inter-contact delays as inter-contact delays contain contact duration in them. The delay associated with the transmission in the very last contact is not however accounted. Therefore, the transmission delay at the very last contact can then be added with the estimated delay. The concerned delay for a given packet depends on the number of packets scheduled to be transferred in the same contact ahead of the given packet and is given by the sum of all such prior transmission times. Considering the packet can be scheduled anytime depending on its priority within the contact duration, this delay would be at most the contact duration of that last contact. Let $c_{\text{last}}$ be the last contact of $\text{path}_k$ and $\gamma(c_{\text{last}})$ be the associated contact duration. Hence, the delay expression for flow $k$ becomes:

$$\text{delay}(k) = \sum_{c_j} \delta(c_j \rightarrow c_{j+1}) + \sum_{c_j} x_j \delta(c_j) + \gamma(c_{\text{last}})$$

For brevity, in the subsequent discussion, we do not use the above equation; instead, we use Equation 12.

### 4.2 Computing End-to-end Transfer Delay

Delay composition algebra is used to compute the end-to-end delay bounds for multi-stage distributed tasks, each executing on a pipeline of resources. In this case, tasks are flows. Individual resources are path links in the virtual connected network mentioned above. Packets are transferred on this links in priority order.
The delay composition algebra maintains so called load matrices, which we refer to as delay matrices. For each vertex of the virtual connected network (i.e., for each contact in the data flow graph), a two dimensional delay matrix of size \( n \times (n + 1) \) is defined, where \( n \) is the number of flows in the network. Semantically, the delay matrix for a certain contact node designates how much delay a higher priority flow imposes on a lower priority flow. Usually the \( k \)-th vector gives the delay associated with flow \( c \). In particular, \( (i, k) \)-th entry of the matrix, denoted by a tuple \((q_{i,k}^c, r_{i,k}^c)\) for contact \( c \), computes the amount of delay that is exerted by flow \( i \) on a lower or equal priority flow \( k \) (for \( i \leq k \)), when packets from both flows wait for the same contact \( c \). The last row, i.e., \((n + 1)\)-th row, contains an additive component, denoted by \( s_k^c \) for flow \( k \), that is added up when contacts are merged by a reduction process. Given all delay matrices for all vertices of the data flow graph, the delay composition algebra then reduces the whole network to a single vertex, being represented by only a single matrix that quantifies the overall delay interactions among all flows in the network.

The algebra uses two reduction operators, namely PIPE and SPLIT: PIPE combines two vertices connected by an edge into one and SPLIT breaks forks into separate chains (for details, see [9]). In Figure 3, we show the reduction process for two flows, \( c_1 \rightarrow c_2 \rightarrow c_4 \) and \( c_1 \rightarrow c_3 \rightarrow c_4 \).

In the following, we describe how the initial delay matrix per contact is computed for our particular data flow graph. Suppose, a packet \( P \) from flow \( k \) is waiting for contact \( c \), where \( \delta(c) \) is the contact period and \( v(c) \) is the contact volume. Since the wait time for the first occurrence of the contact is accounted for separately, we consider only the delay in waiting for returns of the same contact. Hence, the packet experiences zero delay, if it can be forwarded at the very first contact, or \( \delta \) delay if forwarded at the second contact, \( 2\delta \) for third contact, and so on. The total delay is a multiple of \( \delta \), depending on how many returns of the contact it ends up waiting for. This, in turn, depends on the number and size of higher priority packets waiting for the same contact. Let \( \sum_{i=1}^{k} p_i \) be the sum of bytes of all higher or equal priority packets waiting at the contact, where \( p_i \) is the packet size of flow \( i \). Considering the contact can transfer only \( v(c) \) bytes at every occurrence, the delay exerted on \( P \) is given by:

\[
\text{delay}_c(P) = \left[ \frac{\sum_{i=1}^{k} p_i}{v(c)} \right] \times \delta(c) \\
\leq \left( \frac{\sum_{i=1}^{k} p_i}{v(c)} \right) \times \delta(c) \\
= \sum_{i=1}^{k} \left( \frac{p_i}{v(c)/\delta(c)} \right) 
\]

Now, the question arises as to how much delay flow \( i \) imposes on flow \( k \). Obviously, it is \( \frac{p_i}{\delta(c)} \). Adding all such delays imposed by all higher priority packets gives the total delay on flow \( k \) at contact \( c \). This effectively emulates a situation where all packets are waiting in a queue and are then released one after another, in their priority order, at a rate \( \frac{1}{\delta(c)} \). Based on this observation, we can replace a high speed, periodic but discretely available link by a slow but continuous one with the same effective transfer rate. This is called linearization of transfer delay. By this, a contact \( c \) with contact period \( \delta(c) \) and contact volume \( v(c) \) can be replaced by a continuous link of transfer rate \( \frac{v(c)}{\delta(c)} \) (Figure 4). Due to linearization, the estimated transfer delay becomes larger than the actual transfer delay. This over-estimation is however bounded by at most \( \delta(c) \). Since we are computing the worst case delay bound, such over-estimation is allowed as long as the over-estimation is bounded.

Therefore, the initial entries for the delay matrix at contact \( c \) is given by:

\[
q_{i,k}^c = \begin{cases} 
\frac{p_i}{v(c)/\delta(c)} & \text{if } i \leq k \text{ and flow } i, k \text{ pass through } c; \\
0 & \text{otherwise}.
\end{cases}
\]

\[
r_{i,k}^c = 0 \quad \text{and} \quad s_k^c = 0
\]

### 4.3 Computing the Delay Bound

Delay composition algebra gives a reduced system represented by a single delay matrix that quantifies the worst case delay characteristics of the original data flow graph. While computing the equivalent delay quantities, the composition algebra however considers a single packet from each flow. But, there could be multiple packets from each flow in the network (akin to multiple invocations of the same task in a uni-processor system). At this point, we need to convert the flows of the data flow graph into an equivalent task set and let the task set “execute” in a uni-processor setup. As
per the composition algebra, the execution of these tasks in a single processor, if their execution times are appropriately chosen from the final resultant delay matrix, generates the worst case delay bound for the original distributed data flows.

As suggested by the composition algebra, in order to compute delay bound for flow \( k \), we need to build a task set of size \( k \), \( \{ T^*_1, T^*_2, \ldots, T^*_k \} \) such that execution times are chosen as follows:

- For \( i < k \) (i.e., higher priority flows), \( C^*_i = q_{i,k} + r_{i,k} \).
- \( C^*_k = q_{k,k} + r_{k,k} + s_k \).

Now, we run the classical response time analysis [13] to compute the end-to-end delay bound (a.k.a the worst case response time) for each flow. Response time is defined as the time gap between the time when a task appears in the system and the time when it ends, being interrupted by higher priority tasks in the middle. In our case, this response time is actually the end-to-end transfer delay. The response time \( R_k \) of the task \( T^*_k \) is computed by the following recursive relation:

\[
R^{(0)}_k = C^*_k \\
R^{(l+1)}_k = C^*_k + \sum_{i=1}^{k-1} \left[ \frac{R^{(l)}_k}{P_k} \right] C^*_i
\]

where \( P_k \) is the period of the data flow \( k \) for periodic flows, or the least time gap between two successive data packets from the same flow \( k \). The recursion terminates when \( R^{(l)} = R^{(l+1)} \) for some \( l \).

The response time, \( R_k \), gives the upper bound of end-to-end transfer delay, \( D_t(k) \), for flow \( k \). We need to add the precomputed end-to-end contact delay, \( D_c(k) \), (i.e., sum of inter-contact delays along the path) with it to find the total end-to-end delay bound for flow \( k \), as follows:

\[
delay(k) = D_c(k) + D_t(k) \\
\leq D_c(k) + R_k \\
= \sum_{path_k} \delta(c_j \rightarrow c_{j+1}) + R_k
\]

### 4.4 Pessimism in End-to-End Delay

The pessimism in the computed end-to-end delay arises due to two main reasons: smaller inter-contact delay and extended transfer delay.

![Figure 5: Pessimism in end-to-end delay computation](image)

- **Smaller inter-contact delay**: We assumed that once a successful transfer is made, the next contact leading to the next transfer could only happen after an inter-contact delay has been elapsed. Sometime, this inter-contact delay overlaps with the transfer delay. So, the addition simply counts double. For instance, let us consider a flow along the path \( ij \rightarrow jk \) (Figure 5). Let the first contact \( ij \) happen at time 0 and the contact doesn’t lead to successful transfer of a given packet. The transfer happens at the next contact at time 10. The next contact \( jk \) is supposed to happen at time 15. In that case, \( j \) holds the packet for \( 15 - 10 = 5 \) units of time. But, in our computation, we added 15 as the inter-contact delay between \( ij \) and \( jk \). This leads to an over-estimation.

- **Extended transfer delay**: Due to linearization of transfer delay, the transfer delay is over-estimated by an expression \( (x - |x|) \times \delta(c) \), for some \( x \) and contact \( c \) (Equation 13). Since \( 0 \leq x - |x| < 1 \), on the average the over-estimation per contact is \( \frac{1}{2} \delta(c) \). So, the average over-estimation for the entire path is \( \frac{1}{2} \sum_{c \in path(k)} \delta(c) \).

### 5 Evaluation

In this section, we evaluate quality of end-to-end delay bounds for DTNs. We simulate a disaster-response scenario
based on the post-disaster mobility model (PDM) presented in our previous work [29, 30]. PDM [30], implemented on top of the ONE [22] simulator, has four major movement types: inter-center movement (repeated back and forth movement of supply vehicles between relief centers and main coordination centers), rescue worker movement (localized mobility of volunteers in distressed neighborhoods), police patrols (cyclic police patrol movement among neighborhoods), and emergency movement (vehicles attending to an emergency event). There are also several types of fixed nodes including centers, police stations, and hospitals, which act as meeting places of moving nodes that help relay packets among them. All moving entities and centers are equipped with radio devices and these devices run DTN routing protocols. All experiments are conducted for 5 neighborhoods, 2 main centers, 10 relief and evacuation camps, 20 supply vehicles, 20 rescue workers, 5 police patrols, 5 emergency vehicles, and 50 distressed households.

Figure 6 shows the neighborhood map of the scenario. Circles are distressed neighborhoods. The marked line is a police patrol route from one neighborhood to another. The size of the area is $4 \times 4$ km, average vehicle speed 10 m/sec, human speed 1 m/sec, wait time 20-30 minutes.

Figure 6: The disaster mobility scenario. Circles are distressed neighborhoods. The marked line is a police patrol route from one neighborhood to another. The size of the area is $4 \times 4$ km, average vehicle speed 10 m/sec, human speed 1 m/sec, wait time 20-30 minutes.

We construct data flow graphs from a trace file generated for a large number of runs of the above mobility scenario with different parameters. The trace contains delay attributes (mean, min, max, std. dev) of contact intervals and durations for contacts. Contacts are depicted by vertices labeled by the node ID involved in the encounter (e.g. “64:14” denotes a contact between node 64 and 14). Inter-contact delays are labeled on edges between such vertices. In our case, the entire trace contains nearly 1,200 contacts and 30,000 edges. To simplify, we assume that data flows on only parts of that network. Hence, we construct data flow graphs that are subsets of the above trace. Towards that end, we randomly choose a number of paths of certain length (e.g., up to 8) by picking connected edges from the trace. We then pick edges that connect vertices from the one path to another. Figure 7 shows one such data flow graph. Each experiment is conducted on 10 such graphs, each containing 100 random flows. The typical simulation parameters are shown in Table 1.

![Figure 7: The data flow graph containing all flows](image)

As per our network model, we use the worst-case values for contact intervals, contact durations and inter-contact delays to label the vertices and edges in the data flow graph. In Figure 8, we plot the delay distribution of contact intervals, contact durations and inter-contact delays for some 400 contacts and edges of our generated data flow graphs. The graph also shows the corresponding worst case values that have been used by our data flow graphs. For the post-disaster mobility scenario, the average sum of worst case inter-contact delays for a 5-hop path is around 10 hours. That’s why we choose 12-24 hour deadlines for flows.

We evaluate two main performance metrics: delay bound ratio, the ratio between the average delivery delay of packets and the estimated end-to-end delay bound for a particular flow, and utilization per contact, the fraction of time a particular contact transfers packets over its entire contact duration. Bound ratio indicates the tightness of the estimated delay bound compared to actual delivery delay, whereas utilization measures “load” of the network.

<table>
<thead>
<tr>
<th># of flows/graph</th>
<th>Pkt size</th>
<th>Flow period</th>
<th>Deadline</th>
<th>Conf. level</th>
<th>Conf. int</th>
</tr>
</thead>
<tbody>
<tr>
<td>40–100</td>
<td>2.5–10.0KB</td>
<td>0.5–3 Hr</td>
<td>12–24 Hr</td>
<td>95%</td>
<td>1% of mean</td>
</tr>
</tbody>
</table>

Table 1: Simulation parameters
Figure 8: Metrics of vertices and edges of the data flow graph. Delays are shown in ascending order of their average values. The vertical bar shows the 95% confidence interval for the associated metric. The worst case observations, which are used in the data flow graphs, are shown in a separate curve.

Figure 9: Delay bound ratio for varying loads

Figure 9 shows the delay bound ratio for different load scenarios. Figure 9a shows the bound ratio for varying number of flows for different periods. We observe that the bound ratio is approximately 30%, which remains fairly constant as the number of flows increases and the period changes. Figure 9b demonstrates the effect of path length on bound ratio in different deadlines. Deadlines are determined by deadline factors (DF), which means deadline (in hour) per hop. In this case too, the bound ratio remain fairly close to 30%, not being affected much by deadlines and path lengths. This is because factors that affect original packet delay also affect the estimated delay bound in the same scale. That’s why the ratio remains fairly unchanged. In terms of pessimism, the results match closely with the earlier delay composition results, presented in [9].

In order to evaluate the tightness of the estimated delay bound, we plot maximum and average bound ratios for 50 flows in Figure 9c. Maximum bound ratio for a given flow is determined by dividing the maximum packet delay by the estimated delay bound for the same flow. We observe that the average bound ratio remains ≈ 30%, whereas the maximum bound is nearly 60%, sometimes, the bound is even as close as 80%. This high bound ratio is possible when a few packets actually suffer delays that are very close to their estimated bounds. In particular, this can happen to the highest priority flow among a set of competing flows that only undertake the end-to-end contact delay, but zero or very low transfer delay.

We construct an offline (capacity planning) admission controller based on the computed delay bounds. For planning purposes, the admission controller only considers those flows whose deadlines are smaller than the end-to-end delay bound estimated the controller. In a network with that traffic profile, no packets will ever miss their deadlines. The delivery guarantee is however achieved at the cost of reduced utilization of network resources. In Figure 10, we demonstrate the effect of the capacity planning admission controller on network performance. Figure 10a compares when packets are injected into the network in violation of capacity planning and when packets are constrained to what the planning deemed feasible. We observe that in the absence of an admission controller, as more packets are injected, more and more packets miss deadlines. When packets adhere to capacity profiles, fewer packets are used, but all those packets meet their deadlines. Figure 10b plots what fraction of total flows are found admissible for different flow periods and deadlines, compared to no capacity planning.

Figure 10c shows the average utilization per link for different flow periods (1/2 Hr and 1 Hr). Without planning,
the utilization goes up as the number of flows increases, but eventually some flows miss deadlines, which makes the timely traffic lower. When traffic is restricted to computed capacity, utilization goes down, since less flows are found admissible, but all flows meet deadlines.

Figure 10: Effect of admission control. (a) # of packets meets deadline or gets admitted with or without an admission controller, (b) # of flows admitted at various flow periods and deadlines, (c) average link utilization

The estimated end-to-end delay bound is the sum of contact delays and transfer delays. For a given flow, the contact delay is determined by the path of the flow, whereas the transfer delay is due to interactions among flows. Figure 11a shows transfer delays for some 40 flows. In most cases, transfer delays are quite small compared to the end-to-end contact delay, which is, on average, 380 minutes for this flow set. The maximum transfer delay (nearly 470 minutes) is however comparable. Recall that the transfer delay depends on periods of flows (due to Equation 14). Usually shorter periods cause longer response times, hence longer transfer delay. Figure 11b plots the maximum transfer delays for different flow periods. As we observe, the maximum transfer delay decreases as the flow period increases.

We can summarize all results in the following way. The observed end-to-end delay is nearly 30% of the estimated delay bound, irrespective of various parameters. With respect to capacity planning, for a moderate flow period (greater than 1 hour) based on our post-disaster mobility model, nearly 50% and above flows can be admitted so that packets from those flows never miss their deadlines. By using this result, an offline admission controller can rearrange/reroute flows accordingly. We leave constructing an online admission controller to future work.

6 Conclusion

This paper analyzes end-to-end delays of prioritized flows in DTNs using delay composition algebra. Knowing the end-to-end delay bounds for data flows in DTNs can help in planning resource provisioning and even influencing the mobility of agents in practical DTN deployment cases. DTNs pose challenges in characterizing delay attributes of the network, in part, because of their disconnected nature. We model the network as a collection of distributed encounters between nodes and represent sequences of encounters by edges between them. We then systematically convert the DTN model into a “connected” network representation so that data flows are expressed as paths in the graph and the worst case end-to-end delay bounds are computed. Evaluation shows that computed delay bounds are not significantly different from measured worst-case values.
Acknowledgment

This work was funded in part by Fulbright Fellowship IIE-15073255, ONR grant N00014-10-1-0172 and NSF grant CNS 06-26825.

References