

Product Flow Analysis in Distribution Networks With a Fixed Time Horizon

M. T. Wynn C. J. Fidge A. H. M. ter Hofstede M. Dumas

Faculty of Information Technology,
Queensland University of Technology, Australia.
Email: {m.wynn, c.fidge, a.terhofstede, m.dumas}@qut.edu.au

Abstract

The movement of items through a product distribution network is a complex dynamic process which depends not only on the network's static topology but also on a knowledge of how each node stores, handles and forwards items. Analysing this time-dependent behaviour would normally require a simulation algorithm which maintains a globally-synchronised system state. For a certain class of problem, however, where the simulation is required to stop in a consistent state but not necessarily maintain consistency at all times, we show that an algorithm that makes localised decisions only is sufficient. As a motivating example we consider the practical problem of product recalls, in which our primary concern is the state of the distribution network at a specific time after a batch of suspect items was released, but we do not necessarily care about intermediate states leading up to the final one.

Keywords: Network analysis; Product distribution.

1 Introduction

A product distribution network routes items from source nodes to their final destinations, with each item typically passing through several nodes en route. The route that each item takes is controlled by the logistic processes governing each node. Examples include mail systems, small-scale retail networks and large-scale shipping networks. Understanding the behaviour of distribution networks is vital to business process modelling, but is challenging because the dynamic properties of such a network cannot be predicted easily from its static topology. Instead, some form of simulation is usually required to analyse its behaviour.

Such an analysis would normally require us to use an algorithm which maintains a global notion of time throughout the simulated network. Unfortunately, maintaining global states is awkward when dealing with asynchronous distributed systems. Keeping the global state consistent at all points in the computation forces us to allow for the possibility that an action in one node completes mid-way through a concurrent

action in another node. The resulting simulation algorithm thus needs to partition logically atomic steps, adding to its complexity. Also, the algorithm can be inefficient if the time-scale is fine-grained.

We have found, however, that for a certain class of distribution network problem there is no need to maintain a global state. Specifically, when the items moving through the network behave independently of one another, and when the desired result of the analysis is to discover the state of the network at some pre-determined time, we can use an algorithm that does not require synchronisation across the nodes. Instead, we define a global 'horizon' for the calculation and model each item's passage through the network separately until it reaches the horizon. The resulting calculation can be more efficient than the corresponding globally-synchronised simulation algorithm because each item can make progress in comparatively large time steps.

Here we use the highly-topical issue of product recalls as a motivating example to demonstrate the approach. Recent dangerous product recalls in Australia, including several triggered by extortion attempts, have proven enormously difficult and expensive for the companies involved (Safe 2005). Developing a reliable solution to this problem is highly significant: 'over-recalling' is needlessly expensive, whereas 'under-recalling' danger products can cost lives. Finding an efficient solution is also important, given the size of typical product distribution networks (Safe 2005).

Barcodes and RFID tags do not solve the problem of locating goods, because they are useful only after items have been located. Also, recent recalls have involved small, high-volume items, such as chocolate bars, paracetamol tablets, and baked goods (Industry Search 2006). Low-cost items such as these cannot be tracked cost-effectively using satellite technologies. Therefore, when a recall becomes necessary, we must rely on our knowledge of the product distribution network's typical behaviour in order to predict the likely location of suspect items. In this paper we present such an analysis algorithm based around the items themselves.

2 Related Work

Our interest here is an algorithm for analysing the behaviour of a physical distribution network represented as an acyclic directed graph with quantities attached to the nodes and arcs. There are numerous graph analysis algorithms for such capacitated digraphs, including algorithms for calculating shortest paths, minimum spanning trees, maximum capacity along a path, and so on (Berman & Paul 2005, Ch. 11). While relevant, none of these static analysis principles is directly applicable to our problem of identifying a particular state in the dynamic evolution

Acknowledgments We wish to thank the anonymous ACSC reviewers for their helpful comments. This research was funded by Australian Research Council grant DP0773012, *Rapidly Locating Items in Distribution Networks with Process-Driven Nodes*.

Copyright ©2008, Australian Computer Society, Inc. This paper appeared at the Thirty-First Australasian Computer Science Conference (ACSC2008), Wollongong, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 74, Gillian Dobbie and Bernard Mans, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

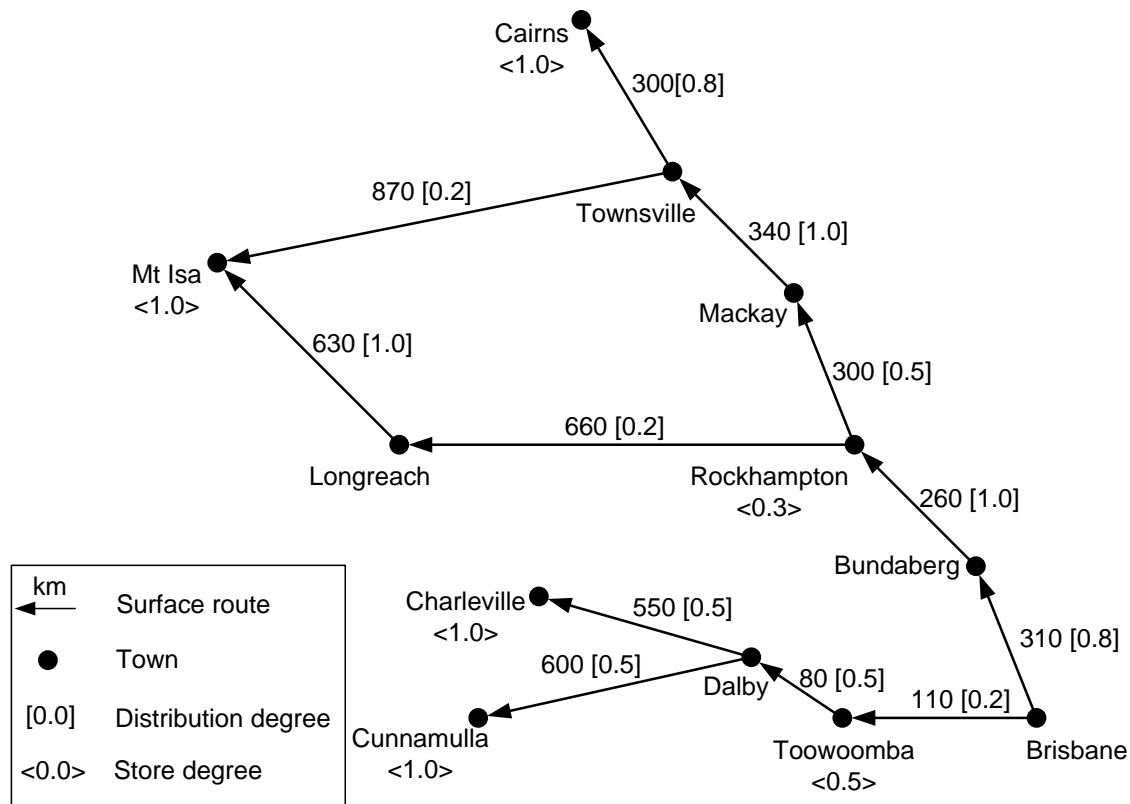


Figure 1: Major Queensland distribution routes from Brisbane

of a distribution network.

With respect to our motivating example of industrial product recalls, there are already several commercial products for managing product recalls in the manufacturing industry. However, these tools are primarily intended for products that are still within the confines of the factory or warehouse (Pulse Logistics Systems 2005, Hajnal et al. 2004), and are normally designed to help isolate the invoices and shipping orders for suspect products. They allow each item's first port of call to be identified, but it is expected that each of these distribution points will then be contacted individually in order to determine what has happened to the items in question (Seradex 2005). Our goal is instead to determine where suspect items are likely to be found after they have left the controlled facility and have started circulating in a physical distribution network.

There is extensive literature in the fields of logistics and supply chain management dealing with the analysis of physical distribution networks (Sarmiento & Nagi 1999). Techniques developed in this field support the design of such networks under various constraints and optimisation criteria like, for example, identifying optimal warehouse locations and vehicle routes (Daskin & Owen 2003). Other problems addressed in this field include performance measurement, such as estimating travel times between locations in a network (Lin et al. 2005). In contrast to this prior work, we do not deal with the design, evaluation or optimization of physical distribution networks. It is assumed that the network is given and that data on its topology, routing behaviour at each node and transportation delays are available. Given these data, we seek to estimate the possible location of products dispatched on a given date, within a fixed time horizon.

A problem closer to the one faced here is that of tracking information flow through electronic communication networks. Prior research on security-critical

communications devices has shown how the potential destinations of classified data can be determined by treating circuitry schematics as digraphs and by modelling the routing of information through each node via the different operating 'modes' of the individual electronic components (Rae & Fidge 2005, Fidge & McComb 2006). A practical tool for information-flow analysis was developed from this theory (McComb & Wildman 2005). Although similar to the problem addressed in this paper, this previous work is distinct because, unlike physical items, 'information' is infinitely copiable. Therefore, this previous information flow algorithm traced all nodes through which information has passed, rather than identifying its 'current' location. Furthermore, this previous work did not incorporate the notion of elapsed time, or of a time horizon, both of which are central to this paper.

3 Motivating Example

As a motivating example we consider the topical problem of locating specific items circulating in large organisations or supply chain networks. This can be a security-critical or safety-critical issue when the item's status changes. For example, rapidly finding a memorandum circulating in a government department can be critical if the document's classification is changed from 'unclassified' to 'secret'. Tracking down a package that has been sent into a postal network is vital if authorities are told it may contain explosive materials. Similarly, locating frozen foods en route to retailers can be a public health issue if they come from a batch found to be contaminated with salmonella. In each of these cases the challenge is to isolate the likely location of particular items which have been released into a largely autonomous, or self-regulating, distribution network.

Consider the situation where a Brisbane-based company must issue an urgent recall for a product that was shipped 48 hours ago for distribution

throughout Queensland. Figure 1 shows all the company’s known transport routes. With the geographical data expressed as a directed graph, it is possible to perform a transitive connectivity analysis to determine where the items of interest *may* have gone. However, this proves to be useless in practice. Assuming an average land speed of 60km/h, the items could have traveled 2,880km since they were released, making it possible to reach any node in the graph. In other words, our conclusion from this simple analysis is that the suspect items could be anywhere! This static analysis fails because it does not take into account how the items are routed at each node and precisely how long they take to travel between nodes.

Now assume that the company has more detailed information about the typical behaviour of each node in the distribution network, including how items are typically stored, handled and forwarded, including any seasonal differences. For example, the arcs in Figure 1 have been annotated with the distribution percentage between each pair of nodes, in square brackets. It shows, for instance, that 80% of all products arriving at Townsville are forwarded to Cairns and the remaining 20% are sent to Mt. Isa. Similarly, the storage percentage associated with a node, in angled brackets, represents the percentage of arriving products offloaded at a particular town for distribution locally. For instance, the figure shows that 50% of all products sent to Toowoomba are consumed in the local region. At sink nodes, i.e., the ends of the distribution chain, we assume that all products are offloaded. For instance, 100% of items arriving at Cairns are offloaded.

In practice this model can be developed and calibrated incrementally over time, using information supplied by distributors based on actual observations of products in the field. (However, if no information is yet available for a particular node a default worst-case model can be assumed. For the product recall problem, this is when the node forwards items on all its outgoing arcs and stores some locally, i.e., it distributes items as widely as possible.)

With this additional information, it is now possible to carry out a more detailed analysis for the possible locations of particular products. Our main concern when a product recall is ordered is to identify those towns at which dangerous goods may be found. This means we are interested in the state of the network at a specific moment in time (usually the present moment) following the release of the items. We call this the ‘horizon’ of the analysis. However, we usually have no interest in how the network reached this state. These characteristics of the problem are central to our algorithm, since they allow us to perform calculations without the need to maintain consistency in the intermediate states leading up to the final one.

4 Basic Product Flow Algorithm

In this section, we describe our algorithm for performing product flow analyses up to a fixed time horizon. We first explain key concepts and the data structures central to the algorithm, and then the individual functions from which the algorithm is composed. Finally, we give a step-by-step explanation of the algorithm’s calculation, using the motivating example from Section 3 above.

4.1 Data Structures

In this section the data structures and access functions which are used by the algorithm are defined.

4.1.1 Network Topology

To perform the analysis, it is critical that the information about distribution routes and connections between different towns is known in advance. A distribution network can be considered as a ‘directed graph’ with vertices representing distribution nodes and edges representing the connections between two nodes. The role of a node in a particular distribution route can vary. A node could represent an initial source of the distribution route (e.g., Brisbane in our sample network), or a final sink node for products (e.g., Cairns in our example), or it could be an intermediate destination for a product, acting as both a target and a source (e.g., Rockhampton). We also assume that the graph of a distribution network is acyclic.

To formalise this notation, let type *Node* represent the distribution nodes in the network graph. A possible connection between two nodes (a source and a target) is defined as an element of relation $Connection \subseteq Node \times Node$. Function $src : Connection \rightarrow Node$ returns the source node of a connection and function $tgt : Connection \rightarrow Node$ returns the target node of a connection.

4.1.2 Distribution Functions

The more detailed information we have about how each node operates, the more reliable the results of the analysis will be. For our purpose, we need to know how a node distributes products to its outgoing connections and what proportion of products are offloaded locally, if any. We use the term ‘distribution degree’ to represent the percentage of products distributed to each connection from a node and ‘store degree’ to represent the percentage of products offloaded at a node.

The distribution degree for a connection is defined as real-valued function $degree : Connection \rightarrow [0 \dots 1]$. The store degree for a node is defined as partial function $storeDegree : Node \rightarrow [0 \dots 1]$, since not all nodes will store items locally. The set of nodes where items could be offloaded is the domain of this function, i.e., $dom\ storeDegree$. Let *Set* be the powerset type constructor, *N* be the set of natural numbers, and type $NodeStorage = Node \times N$ comprise node-number pairs. Then function $split : Set(Connection) \times N \rightarrow SetNodeStorage$ makes use of the *degree* and *storeDegree* functions to return a set of node-number pairs that represents the number of items to be sent to a given node. This function ensures that the total number of products forwarded to various nodes is the same as the total number received.

4.1.3 Packages

We use the term ‘package’ to describe a parcel of products that is being distributed on a network route. We assume that a package contains a number of items of a given product. It is possible to split a package into two or more non-empty packages, preserving the total number of items, that can be then distributed further in the network. The route that a package follows is solely dependent on the logistical behaviour of the network nodes (i.e., packages are not individually addressed).

In the algorithm we keep track of the number of items in each package, the time the package arrived at its current (or last) node, the location of the package (the source and target of the last distribution step) and whether or not a package has reached its final destination. Let *Package* represent the type of a package that is routed across a network. The

Function DistributePackage**Input:** $p : Package$ **Output:** $P' : SetPackage$ **Pre:** $\neg finalDestination(p)$ **begin** $SetConnection\ NextConnections := \{c : Connection \mid tgt(location(p)) = src(c)\};$ $SetNodeStorage\ SplitCount := split(NextConnections, count(p));$ **for** $c \in NextConnections$ **do** $Package\ p' := new\ Package();$ $timeStamp(p') := timeStamp(p) + delay(c, p);$ $location(p') := c;$ $count(p') := SplitCount(tgt(c));$ $finalDestination(p') := false;$ $P' := P' \cup \{p'\};$ **end** **if** $tgt(location(p)) \in dom\ storeDegree$ **then** $Package\ p' := new\ Package();$ $timeStamp(p') := timeStamp(p);$ $location(p') := location(p);$ $count(p') := SplitCount(tgt(location(p)));$ $finalDestination(p') := true;$ $P' := P' \cup \{p'\};$ **end** **return** P' ;**end**

functions associated with a package are as follows: $count : Package \rightarrow N$ returns the number of items in a package, $timeStamp : Package \rightarrow N$ returns the absolute time delay (in hours) since a package started its journey, $location : Package \rightarrow Connection$ returns the current location of a package, which is interpreted to mean that the package is either en route on this connection or has reached the connection's target node, and $finalDestination : Package \rightarrow Boolean$ returns whether or not the package has reached its final destination.

The propagation delay, in hours, of a package traversing a connection is determined by function $delay : Connection \times Package \rightarrow N$. For now, we treat this function as a simple mapping from connections to fixed delays. In Section 5, we demonstrate how this function can be made more realistic by introducing the notions of package handling delays, transport delays, and lead time delays.

4.1.4 Time Horizons

The time horizon indicates when the analysis should terminate and it is an important feature of our analysis technique. The intention is to calculate the way packages move through the network until a predetermined time is reached. At this time it is possible that all packages have arrived at their final destinations and the network is quiescent. Alternatively, only some of the packages may have reached their final destinations and others may be still en route. This latter situation is the challenge of particular interest to us. In effect, we want the calculation to stop in a state which gives a consistent snapshot of where each package is in the network when the time horizon is reached. The horizon itself is provided as an input to the algorithm and it is represented as an integer denoting elapsed hours.

4.1.5 Network State

The algorithm's state represents the status of packages at intermediate steps during their journey as well as the status of the various packages at the final state, when the analysis reaches a given time horizon. The

state is made up of three sets of packages, as explained below.

The analysis stops when the predetermined time horizon is reached by all packages. When completed the calculation returns the following three sets of packages:

- **Final:** This set represents all the packages that have reached their final destinations before or at the given time horizon. If all packages have reached their final destinations, the other two sets will be empty.
- **Leading edges:** This set represents all the packages that are in transit, i.e., traversing a connection, when the time horizon is reached. It includes the (future) times at which they will reach their next node.
- **Trailing edges:** This set also represents the packages that are in transit when the time horizon is reached. However the additional attribute in this set is the (past) time at which each package left its previous node.

To support this, the algorithm's data structure *State* is declared as $SetPackage \times SetPackage \times SetPackage$, in which function *final* returns the first set of packages, function *leading* returns the second set of packages, and function *trailing* returns the third set of packages in a state.

4.2 Baseline Algorithm

Here we describe the three functions that make up the basic algorithm for product flow analysis.

- **DistributePackage:** This function describes how a package could be split into child packages and distributed to the connected nodes in one step using the distribution functions available for each node. Function *DistributePackage* (above) takes a package as input and returns a set of packages that are to be distributed in the next step.
- **ProgressPackages:** This function contains the program logic whereby the timestamp of a package is compared with the given time horizon to

Function ProgressPackages**Input:** $packages : SetPackage, horizon : N, sn : State$ **Output:** $state : State$

```

begin
  state := sn;
  for  $p \in packages$  do
    for  $p' \in DistributePackage(p)$  do
      if  $timeStamp(p') \leq horizon \vee finalDestination(p')$  then
         $final(state) := final(state) \cup \{p'\}$ ;
      end
    else
       $leading(state) := leading(state) \cup \{p'\}$ ;
       $Package p_t := new Package()$ ;
       $timeStamp(p_t) := timeStamp(p)$ ;
       $location(p_t) := location(p)$ ;
       $count(p_t) := count(p')$ ;
       $finalDestination(p_t) := finalDestination(p')$ ;
       $trailing(state) := trailing(state) \cup \{p_t\}$ ;
    end
  end
end
return state;
end

```

Function PerformFlowAnalysis**Input:** $I : SetPackage, horizon : N$ **Output:** $state : State$

```

begin
  State state := new State();
  SetPackage P := I;
  state := ProgressPackages(I, horizon, state);
  while  $P \neq final(state)$  do
     $P := final(state)$ ;
     $final(state) := \emptyset$ ;
    state := ProgressPackages(P, horizon, state);
  end
  return state;
end

```

determine how the state of the analysis should be updated with the information about the child packages. If the timestamp of a child package is less than or equal to the horizon, the package is put back into the final set for further distribution and/or storage. Otherwise, the package is stored in the leading and trailing sets. This indicates that the package is in transit and it has left the source at the time recorded in the trailing set and it should arrive at the target node at the time recorded in the leading set. Function *ProgressPackages* (above) takes as input a set of packages, produces a number of children packages for distribution (in one step) and then returns the results based on the given horizon. This function makes use of the *DistributePackage* function.

- *PerformFlowAnalysis*: This is the main function that calls the *ProgressPackages* function iteratively until the final state is reached for a given horizon. Function *PerformFlowAnalysis* (above) performs a fixed-point calculation and stops when there is no change in the final set of packages in the state. The function takes as inputs the initial set of packages to distribute and the time horizon to stop the analysis.

To validate this algorithm, we have developed a prototype implementation (downloadable from <http://www.yawlfoundation.org/research/simulation.php>).

4.3 Illustration

Here we illustrate the functioning of the algorithm using our earlier motivating example (Section 3). Consider the scenario where a Brisbane-based company must issue an urgent recall for a product which was shipped as a package containing 1000 items 48 hours ago for distribution as per the network shown in Figure 1.

Using the data given in Figure 1, regarding the distances between nodes and the way items are routed, we calculated where the products could be after 48 hours. Figure 2 shows the ‘final state’ information from the analysis overlaid with the network graph. The algorithm starts with a package of 1000 items, with timestamp ‘0’ at Brisbane and horizon of ‘48’. The sample distribution degrees and store degrees for each node are then used to determine the appropriate splitting of packages. For instance, from Brisbane a package with 800 items (due to the 0.8 degree assigned to the connection Brisbane-Bundaberg) is sent to Bundaberg and a package with 200 items is sent to Toowoomba (due to the 0.2 degree assigned to the connection Brisbane-Toowoomba). The package arrives at Toowoomba after a certain delay, in this case 7 hours from the time it left Brisbane. The timestamp represents the arrival time of the package at a node and it is represented using the “@” symbol. At Toowoomba, a package with 100 items is stored (due to the 0.5 store degree assigned to the node Toowoomba) and a package with 100 items is

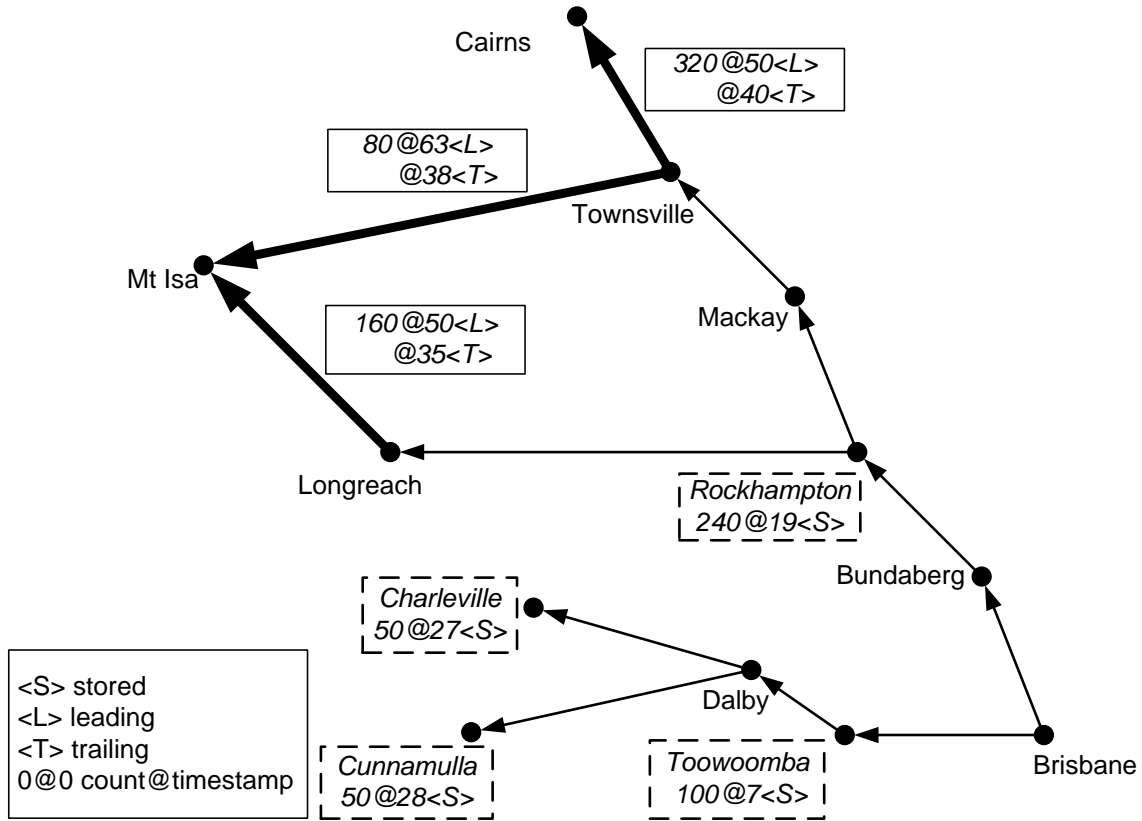


Figure 2: Possible locations of 1000 items distributed from Brisbane 48 hours ago

forwarded to Dalby. From Dalby, the packages are distributed to Charleville and Cunnamulla as shown in the figure. As the example network shows Cunnamulla and Charleville as final destinations, all items are offloaded at these destinations and marked as ‘stored’ (dashed boxes in the diagram). For other packages, the distribution continues until the given horizon is reached. From the figure, one can see that three packages are still en route after 48 hours has elapsed (solid boxes in the diagram with bold arrows for connections). These packages are shown with the trailing and leading times. For instance, there is a package containing 160 items travelling between Longreach and Mt. Isa. The package is expected to arrive at Mt. Isa at time 50 and it left Longreach at time 35. Similarly, there is another package containing 80 items due to arrive at Mt. Isa at time 63 from Townsville. This snapshot information is very useful to determine not only where items could be located but also to eliminate locations where items could not be.

5 Computing Delays

The basic algorithm uses a simple delay function to determine how long it takes a package to get from one node to the next. In this section, we show how to compute a more realistic delay duration by taking into account the transport delay based on the distance between connections, the transport mode and its speed, the handling delay at each node based on the availability of resources, and the lead time delay based on the operating hours of a distribution node.

To do this the delay function is extended to take into account the transport delay, the handling delay, and the lead time delay as follows.

Function delay

Input: c : Connection, p : Package
Output: delay : N
begin
 Node $n := \text{tgt}(\text{location}(p))$;
 Day $d := \text{day}(\text{arrivalDayTime}(p))$;
 Time $t := \text{time}(\text{arrivalDayTime}(p))$;
 delay := transportDelay(c) +
 handlingDelay(n , count(p)) +
 leadtimeDelay(n , d , t);
return delay;
end

We first describe how the transport delay can be calculated based on the various modes of travel and corresponding speed, and how the handling delay can take into account the employee’s work rates at each node. (Of course, further data about transport speeds and handling rates could be incorporated into the algorithm in practice.) To do this we assume that the following additional functions are available for a connection: $\text{distance} : \text{Connection} \rightarrow N$ and $\text{speed} : \text{Connection} \times \text{Mode} \rightarrow N$, where type $\text{Mode} = \{\text{air}, \text{surface}\}$. The transport delay can be then calculated as follows:

$$\text{delay} := (\text{integer})\text{distance}(c) / \text{speed}(c, \text{mode}(c)).$$

Further, assume that the following additional functions are available for a node: $\text{empNum} : \text{Node} \rightarrow N$ which returns the number of employees and $\text{rate} : \text{Node} \rightarrow N$ which returns the average number of items that could be handled by an employee per hour. From this information, the handling delay can be calculated as follows:

$$\text{delay} := (\text{integer})\text{items} / \text{rate}(n) * \text{empNum}(n).$$

We can also introduce a realistic lead time delay function either based on a weekly schedule or calen-

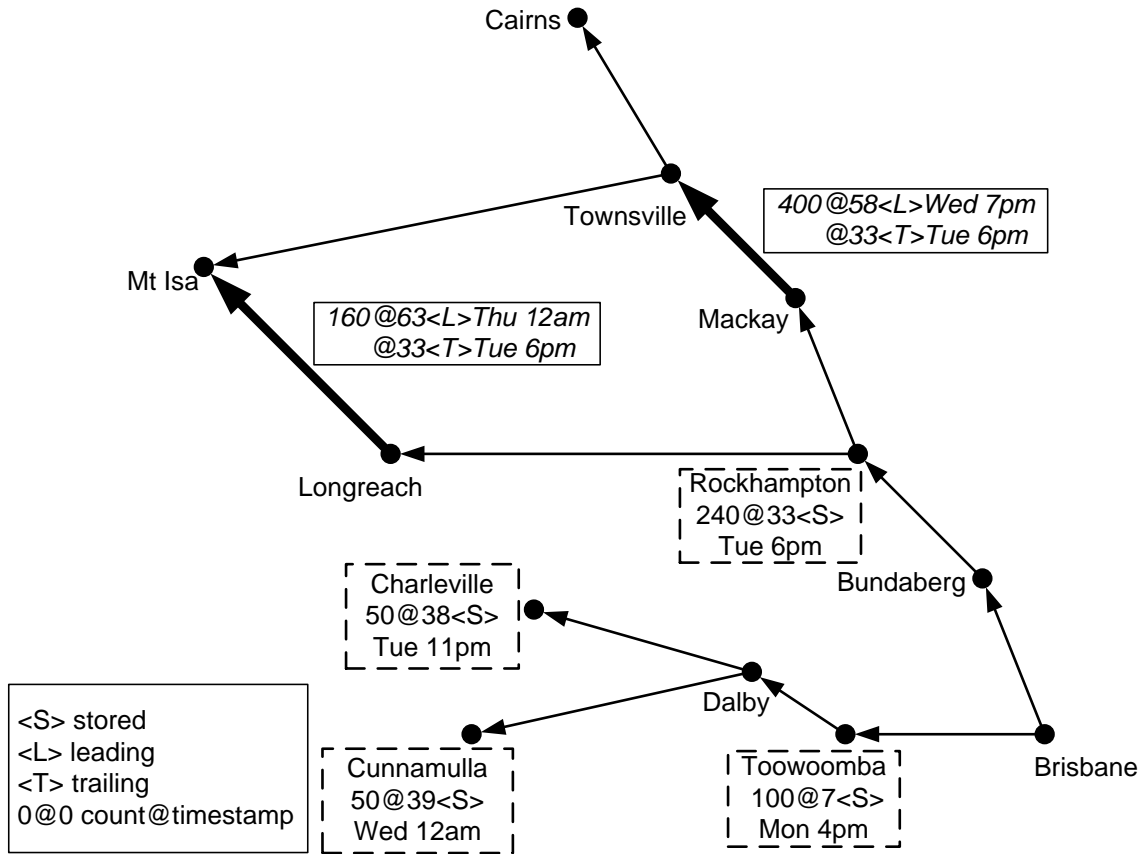


Figure 3: Locations of 1000 items 48 hours after being distributed from Brisbane on Monday at 9am

dar time. We base the weekly schedule on the known operating hours of each distribution node. For instance, a distribution centre may be open between 9–5pm Monday to Friday and 10–4pm on Saturdays but is closed on Sundays. In addition, we assume that the timestamps associated with packages are based on calendar, rather than elapsed, time. For instance, we may know that a package was sent on a Monday morning at 9am.

We can then determine where items from that package are in 48 hours time by taking into account the operating hours of the distribution nodes along the way. In this situation the location of the items could vary significantly due to effect of distribution nodes being either open or closed when a package arrives.

Let data type *Day* represent days of the week [Mon...Sun], and *Time* represent hours of the day [00...23]. Constructor *DayTime* : *Day* × *Time* returns the combination of the two, and selectors *day* : *DayTime* → *Day* and *time* : *DayTime* → *Time* extract the component parts of a timestamp. Let function *next* : *Day* → *Day* return the next day of the week, and function *addHours* : *DayTime* × *N* → *DayTime* return the new day and time after adding the number of hours given as *N*, both using circular arithmetic.

Also let functions *SB* : *Node* × *Day* → *Time* and *CB* : *Node* × *Day* → *Time* return the start of business and close of business times of a distribution node, respectively. Function *DayOff* : *Node* × *Day* → *Boolean* returns whether a distribution node is closed on a certain day. New function *arrivalDayTime* : *Package* → *DayTime* returns the arrival day and time of a package.

We can now define function *leadtimeDelay* to take as inputs a distribution node and a day and a time (which represent the arrival day time of a package)

and return the delay in hours. The resulting delay is calculated based on the arrival time of the package, the number of days that the distribution node is closed and the opening hours of the next business day.

```

Function leadtimeDelay
Input: n : Node, d : Day, t : Time
Output: delay : N
begin
  delay := 0;
  if t ≥ CB(n, d) then
    delay := 24 - t;
    while DayOff(n, next(d)) do
      delay := delay + 24;
      d := next(d);
    end
    delay := delay + SB(n, d);
  end
  return delay;
end

```

This delay calculation, including a weekly calendar, has been implemented in the prototype analysis program. In the calculations, we assume that it is possible for a package to arrive at any point in time (even during the closing hours). However, package are only forwarded to their next destination during opening hours.

As an example, Figures 3 and 4 provide a comparison of two calculations involving packages of the same size (1000 items) and the same fixed time horizon (48 hours). The difference between them is the day of the week on which the package is assumed to have started its journey. From Figure 3 it can be seen that if the package is sent on Monday at 9am, some of the items will reach their final destinations of Charleville and Cunnamulla before the expiry of the 48 hours time horizon. Other items will be in transit

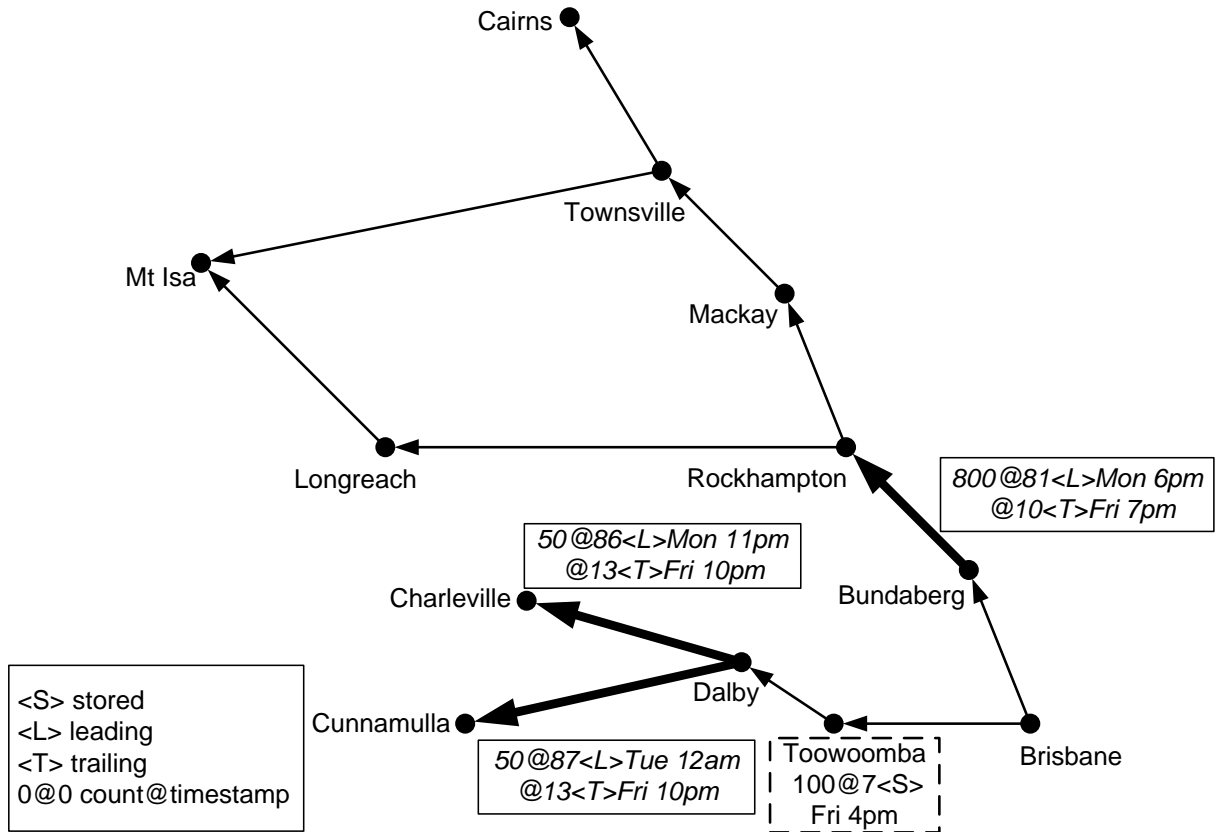


Figure 4: Locations of 1000 items 48 hours after being distributed from Brisbane on Friday at 9am

beyond Longreach and Mackay. However, Figure 4 reveals a quite different situation if the package is sent on Friday morning. Thanks to the processing delay introduced by the weekend, no items will have reached Charleville or Cunnamulla when the time horizon is reached, nor will any have gone as far north as Rockhampton. Being aware of the dramatic difference between these two outcomes would have a major impact on the speed with which a recall of dangerous goods could be conducted.

Another alternative extension to the basic algorithm is to take into account the actual date that a package has been sent, e.g., 24/12/2007, rather than the weekday. Function *leadtimeDelayDate* (below) describes how the delay can be calculated using a calendar date.

Function <i>leadtimeDelayDate</i>
Input: $n : Node, d : Date, t : Time$
Output: $delay : N$
begin
$delay := 0;$
if $t \geq CBDate(n, d)$ then
$delay := 24 - t;$
while $Holiday(n, nextDate(d))$ do
$delay := delay + 24;$
$d := nextDate(d);$
end
$delay := delay + SBDate(n, d);$
end
return $delay;$
end

Here data type *Date* is $Day \times Month \times Year$ representing a date such that *Day* is in range $[0..31]$, *Month* is in $[0..12]$ and *Year* is a 4-digit number. Let function $nextDate : Date \rightarrow Date$ return the next day. Similar to the functions defined for *DayTime*, we can now define a set of functions for calendar dates that take into account the opening and

closing times for a particular date as well as public holidays. To do this we would let partial functions $SBDate : Node \times Date \rightarrow Time$ and $CBDate : Node \times Date \rightarrow Time$ return the opening and closing times respectively of a distribution node; partial function $Holiday : Node \times Day \rightarrow Boolean$ return whether or not a distribution node is closed on a certain date; and define functions $arrivalDate : Package \rightarrow Date$ and $arrivalTime : Package \rightarrow Time$.

6 Conclusion

Simulating the behaviour of a communications or transport network is normally a complex and challenging computation, requiring maintenance of a consistent global state across all nodes. However, we have identified an algorithm for analysing such behaviours that relies on local states only, provided that objects move through the network independently, and our overall goal is to determine their location at a pre-determined time. The algorithm determines the way each item, or package of items, moves through the network, but achieves a consistent global state only when all items reach the fixed time horizon.

The usefulness of the algorithm was illustrated by showing how it can be applied to the problem of product recalls in distribution networks. It was also shown how the basic algorithm could be made more realistic by, for instance, incorporating a notion of calendar, rather than just elapsed, time.

Finally, we note that the algorithm described in this paper can be extended further in a number of ways:

- Rather than showing how a batch of items is split-up and distributed throughout the network, the probability that a particular single item arrives at different destinations can be calculated merely by changing the way we interpret the ‘dis-

tribution degree'. Instead of representing a percentage split of items, it can be viewed as a percentage probability that the item of interest follows a particular route. The number at each node when the calculation ends then denotes the probability that the item is there at the time horizon.

- The functions for accounting for the handling and transport times could be made more realistic by changing them from simple numbers into probability distributions, based on observed or measured data. (In this case we would want to associate confidence intervals with the items' locations.)

A more dramatic extension would be to relax the assumption that items do not interact with one another. Doing this would require basing computational steps around nodes rather than packages. Nevertheless, it may be useful to introduce this concept to allow, for instance, modelling of the situation where a delivery truck will not depart from a node until enough items have arrived to fill it.

References

- Berman, K. A. & Paul, J. L. (2005), *Algorithms: Sequential, Parallel and Distributed*, Thomson. ISBN 0-534-42057-5.
- Daskin, M. & Owen, S. (2003), Location models in transportation, in R. Hall, ed., 'Handbook of Transportation Science', Kluwer, pp. 321–370.
- Fidge, C. J. & McComb, T. (2006), Tracing secure information flow through mode changes., in V. Estivill-Castro & G. Dobbie, eds, 'Computer Science 2006, Twenty-Ninth Australasian Computer Science Conference (ACSC2006)', Vol. 48 of *Conferences on Research and Practice in Information Technology*, Australian Computer Society, pp. 303–310.
- Hajnal, E., Kollar, G. & Lang-Lazi, M. (2004), 'IT support and statistics in traceability and product recall at food logistics providers', *Periodica Polytechnica Chemical Engineering* **48**(1), 21–29.
- Industry Search (2006), 'Forty lines of Top Taste products are on recall nationwide', <http://www.industrysearch.com.au/news/printarticle.asp?id=20402>. Accessed June 2006.
- Lin, H.-E., Zito, R. & Taylor, M. (2005), 'A review of travel-time prediction in transport and logistics', *Proceedings of the Eastern Asia Society for Transportation Studies* **5**, 1433–1448.
- McComb, T. & Wildman, L. (2005), SIFA: A tool for evaluation of high-grade security devices., in C. Boyd & J. M. G. Nieto, eds, 'Information Security and Privacy, 10th Australasian Conference (ACISP 2005)', Vol. 3574 of *Lecture Notes in Computer Science*, Springer, pp. 230–241.
- Pulse Logistics Systems (2005), 'Product recall software added', http://www.pulse.com.au/news_detail.asp?NewsID=55&Source=News. Accessed January 2005.
- Rae, A. & Fidge, C. (2005), 'Information flow analysis for fail-secure devices', *The Computer Journal* **48**(1), 17–26.
- Safe, M. (2005), 'Mars attack', *The Weekend Australian Magazine*, 10–11 September pp. 18–23.
- Sarmiento, A.-M. & Nagi, R. (1999), 'A review of integrated analysis of production-distribution systems', *IIE Transactions* **31**(11), 1061–1074.
- Seradex (2005), 'Product recall—Seradex ERP module', http://www.seradex.com/Product_Recall_Module.shtml. Accessed January 2005.