TimelyWare: a Middleware for Timely and Reliable Communication over the Internet

Helton Luis Semedo Craveiro Miranda

Thesis to obtain the Master of Science Degree in
Telecommunications and Informatics Engineering

Supervisor: Prof. Miguel Pupo Correia

Examination Committee
Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Miguel Nuno Dias Alves Pupo Correia
Member of the Committee: Prof. Alysson Neves Bessani

May 2015
For my family.
Acknowledgments

First I want to thank my supervisor for sharing his experiences and knowledge throughout the year. Also I want to thank him for being available every time that I needed and without his help this thesis would not be concluded.

I also want to thank to all my friends for the full support while enduring this process and for believing in me in every step of my studies. Also I want to thank my colleagues for the knowledge that was shared during my studies in the University and for providing relevant opinions about this work.

Finally I want to send a special gratitude to my family for keep pushing me to finish the work even in the difficult times. In general I want to thank them for supporting me throughout all my studies at University even being very distant physically.
Resumo

A entrega de mensagens a tempo é um problema importante por exemplo no contexto das infraestruturas críticas e de cloud, nas quais alguns comandos remotos têm de ser executados dentro de um período de tempo limitado. Este trabalho aborda o problema de assegurar a entrega de mensagens a tempo e de forma fiável em redes de larga escala como a Internet. É considerada apenas mensagens pequenas como por exemplo as mensagens de controlo. O trabalho propõe o sistema TimelyWare, um middleware para entregar mensagens a tempo com uma alta probabilidade explorando os conceitos de redes sobrepostas e de multihoming. O middleware atinge este objectivo adaptando-se ao estado da rede e fazendo predição de estados futuros. O artigo apresenta um algoritmo para fazer essa predição através da extração de padrões temporais. É apresentada uma implementação do sistema e a sua avaliação experimental efectuada no PlanetLab.

Palavras-chave: Adaptação, Middleware, Predição de eventos, Entrega atempadamente
Abstract

Delivering messages within a deadline is an important objective in several cloud infrastructures in which remote instructions have to be executed within limited time. This work addresses the problem of delivering messages on time over wide-area networks such as the Internet, considering just small messages. We present a middleware called TimelyWare that aims to deliver messages in a timely way with higher probability than alternative solutions. This is possible by exploring the concepts of overlays networks, multihoming, and prediction of timing failures. This system was implemented with the goal of not needing to change any of the network infrastructures in order to work with new and legacy systems. The paper presents the middleware routing protocol and the prediction algorithm. It also presents an experimental evaluation using PlanetLab.

Keywords: Event Prediction, Timely Delivery, Dependability, Overlay Network
# Contents

Acknowledgments ......................................................... v
Resumo ........................................................................... vii
Abstract ........................................................................... ix
List of Tables ........................................................................ xiii
List of Figures .......................................................................... xv
Nomenclature ........................................................................... 1
Glossary ................................................................................ 1

1 Introduction ................................................................. 1

2 Objectives ........................................................................... 3

3 Related Work ............................................................... 4
   3.1 Timely Communication in Wide Area Networks .................. 4
      3.1.1 ATM ................................................................. 4
      3.1.2 QoS in Distributed Multimedia Applications over the Internet . 5
      3.1.3 Overlay Networks .................................................. 7
      3.1.4 Multihoming ........................................................ 9
      3.1.5 Data Center Networks ............................................. 12
      3.1.6 Discussion .......................................................... 13
   3.2 Network Parameters Estimation ...................................... 14
      3.2.1 Round-Trip Delay ................................................. 14
      3.2.2 One-Way Delay ................................................... 15
      3.2.3 Available Bandwidth ............................................ 16
      3.2.4 Packet Loss ........................................................ 17
      3.2.5 Path Diversity ...................................................... 17
   3.3 Dependability in Message Delivery ................................. 18
      3.3.1 Dependable Adaptation in Uncertain Environments .......... 18
      3.3.2 Network Channel State Prediction ............................. 19
   3.4 Summary ....................................................................... 23
# List of Tables

3.1 Comparison of the existent solutions according to the proposed requirements .......................... 14  
3.2 Comparison between Delay estimation tools. ........................................................................ 24  
3.3 Comparison between Available Bandwidth estimation tools. ................................................ 24  
3.4 Comparison between Packet Loss Rate estimation tools. ..................................................... 24  
3.5 Comparison between Path Diversity estimation tools. .......................................................... 25  

5.1 Measurement frequency for network parameters. ................................................................. 38  
5.2 Success rate and channel usage ratio for each strategy in a stable environment. ................. 40  
5.3 Success rate and channel usage ratio for each strategy in an unstable environment. ............ 41
List of Figures

3.1 Overlay Architecture example ........................................ 7
3.2 Multihoming example ..................................................... 10
3.3 Online Failure Prediction definition .................................... 20

4.1 TimelyWare system network .............................................. 27
4.2 TimelyWare Architecture .................................................. 28
4.3 Delay Measurement Process .............................................. 30
4.4 RTT samples collected over a week ................................... 32

5.1 Number of msgs sent per attempt by each strategy ................ 40
5.2 Number of msgs sent per attempt by each strategy ................ 42
Chapter 1

Introduction

Many systems that were considered unsuitable for the Internet are now using it. The reason for this change is probably that it reduces the costs involved, by avoiding building and managing private networks. One example are critical infrastructures such as the power grid. Although their communications are quite critical, they are often interconnected by virtual private networks that share the same resources as normal Internet traffic [40]. Another example are cloud infrastructures, whose data centers are often connected over the Internet, even for control traffic.

Some of these systems require messages to be delivered on time, i.e., before a certain deadline. Normally they rely on protocols that can assure these requirements most of the time, e.g., TCP or UDP over IP. Although some of the systems are able to support violations in the requirements, there are those that have stringent requirements. For instance, for power grid control deadlines of 2s, 1s, or less are common [34].

However, meeting deadlines in wide area networks (WANs) such as the Internet is difficult, as their dimension and number of users leads inevitably to time unpredictability. Moreover, timeliness in WANs is susceptible to events such as component failures [49, 22], congestion, and distributed denial of service (DDoS) attacks [62]. These events not only delay communication, but also make this delay unpredictable.

There are some solutions that aim at delivering messages faster in the Internet. For instance, overlay networks [16, 17, 75] and multihoming [12, 19] have been used to allow routing traffic through different paths at the application layer, without modifying the network infrastructure. However, the objective of these works is to deliver messages faster, not to meet deadlines. Moreover, most of these works consider what we call small messages, i.e., messages carried in a single packet and with a negligible length.

The objective of this work is to provide TimelyWare, a middleware that timely delivers messages over WANs in a reliable way with higher probability than alternative solutions. This is possible by using indirect channels, i.e., logical connections between two nodes obtained using an overlay network, besides the ones that the network operator provides, and by monitoring these channels. TimelyWare is also cost sensitive, meaning that we aim avoiding sending unnecessary messages, as done by approaches like flooding the network. This is done by using mechanisms that detect and adapt to the current network
state (congestion, delay, bandwidth), including prediction of timing failures. The motivation for this mechanism is our observation that there are network issues that impact timeliness but happen periodically, so they are predictable. TimelyWare works at application layer, so it does not need modifications to the underlying network in order to run.

This solution combines overlay networks with multihoming routing strategies to assure that messages are delivered within the deadline. The main novelty is TimelyWare’s ability to adapt to the current state of the environment and predict problematic periodic events. The goal is specifically to predict when a channel will change into a problematic state, so that alternative channels can be used instead.

The contributions of this work are: (1) an algorithm to predict channel problems; (2) a protocol for dynamically selecting a set of channels for small messages to be delivered on time over a WAN; (3) a second protocol for large messages.

This report is organized as follows. Chapter 2 briefly describes the goal, the requirements and the expected results for this work. Chapter 3 presents an overview of solutions that are related to this work. Chapter 4 presents the architecture and implementation details of the system and Chapter 5 describes the results obtained in the evaluation phase. Finally Chapter 6 presents the respective conclusions.
Chapter 2

Objectives

This work addresses the problem of providing message latency and reliability assurances for control and file transfer traffic in WANs.

The goal of this work is to design and implement a practical solution to achieve timely and reliable communication with high probability in WANs, such as the Internet, at application level without requiring any changes in the network.

Therefore the proposed solution should have the following requirements:

- It should deliver the messages in a timely manner;
- It should provide that property small messages;
- It should provide reliability in the delivery assuring that the message will be delivered according to some probability;
- It should be cost sensitive, avoiding sending non-necessary packets like flooding.

This work will produce the following results: i) specification of the algorithms for delivering the message within the deadline in a reliable way; ii) implementation of a prototype of this system; iii) extensive experimental evaluation with the prototype.
Chapter 3

Related Work

The goal of the majority of the solutions in this field is to improve the system availability and performance in terms of delay or bandwidth consumption in WANs. Others approaches rely on dedicated networks to link their facilities in order to provide the same goals. This section addresses these approaches in order to understand what are the main challenges for this work.

Section 3.1 presents the existent solutions to obtain timeliness and reliability assurances. In the end it presents a summary of the solutions describing their advantages and disadvantages. Section 3.2 describes some algorithms that are used to estimate the following network parameters: latency; available bandwidth; packet loss; and path diversity. Section 3.3 addresses the algorithms for adapting timeout values and predicting the network channel state. These type of algorithms improve the reliability assurance. The last section presents a discussion about these works comparing them and showing which algorithms are the most suitable for this work.

3.1 Timely Communication in Wide Area Networks

Today there is a significant number of applications that require certain level of quality of service (QoS) in WANs. The QoS is specified in terms of throughput, bandwidth, packet loss ratio and latency. However todays WANs can not provide these QoS requirements because it is best effort. Therefore applications rely on techniques to maintain the required degree of QoS. This section addresses these techniques describing their advantages and disadvantages taking into consideration the requirements for this work.

3.1.1 ATM

One of the firsts approaches to deliver QoS in WANs was Asynchronous Transfer Mode, designated as ATM [25, 33]. ATM is a transmission, packet switching and multiplexing technique designed for a network that support both high-throughput data traffic and real-time, low-latency content such as voice and video. In [61] it was described that this technique can take advantage of bursty services while guaranteeing acceptable performance for continuous-bit-rate services. This is possible because ATM splits the usable
bandwidth into small and fixed size units designated as cells and these cells are allocated to services on demand.

ATM defines four type of QoS classes and each one establishes the required performance and traffic parameters like cell loss ratio (CLR) or cell delay variation (CVD). The first class, the default class or stringent class, requires small buffers with 100 cells. In this class the CLR has to be very low, almost to zero. The second class, denominated as tolerant class, requires large buffers of at least 1000 cells. The third class, bi-level, also requires small buffers, however the maximum CLR required is higher than in the stringent class. The last class, U class, has no requirements.

In order to optimize the selection of the performance and traffic parameters ATM provides multiple services categories which are CBR, rt-VBR and nrt-VBR, ABR and UBR. CBR, constant bit rate, is used in connections that need fixed rhythm. Rt-VBR, real time variable bit rate, is used for applications sensible to delays like video or voice. Non real time variable bit rate or nrt-VBR is used for applications which generate traffic bursts that are not sensible to delay. UBR, named as unspecified bit rate, is a best effort type of service which is similar to what the Internet provides. Finally ABR, designated as available bit rate, is used for applications that can adapt their traffic accordingly to the network conditions.

Although ATM can support different type of traffic classes thus providing the required QoS to applications, it is a complex framework. One of the main reasons for not being massively used was the difficulty of deploying its interfaces to end hosts applications. Another issue was the high cost of the ATM network adapter in comparison with the Ethernet network adapter as referenced in [54]. Due to these issues ATM nowadays is used mainly in the core of the networks because of its unique advantages in providing the required QoS.

3.1.2 QoS in Distributed Multimedia Applications over the Internet

Multimedia application is a combination of multiples forms of communications like text, sounds, videos or graphics. Most of these applications like voice over IP (VOIP), television over IP (IPTV) or video on demand (VOD) are sensitive to end-to-end latency because if it is higher than a certain value the users may notice it. Nevertheless, latency is not the only network parameter that affects multimedia applications. This issue leads to a bad user experience with the application which is an important factor in this case. Therefore they usually require a minimum QoS in terms of packet loss, latency and others parameters from the underlying network. Recently most of this type of applications are being executed over the Internet in order to reduce their costs like IPTV for example. However the Internet provides only best effort services, so these applications have to employ techniques that help to assure the required QoS. Nevertheless these mechanisms do not succeed every time. This subsection is going to address some of these techniques that are used.

As the Internet itself does not support QoS requirements there were proposed some techniques to expand it so that it could support multimedia applications for example. [36] described the following techniques: admission control, policing, priority scheduling of packets and isolation. Admission control is a technique that compares the service QoS requirements and the available resources and based on that
decides whether to accept or reject the service request. Another technique is policing which determines if a packet is obeying the service level agreement (SLA) negotiated between the origin and the network and if not it may drop that packet. Routers employs scheduling algorithms that share the available bandwidth in a fair manner or taking into consideration the priority of the variety of traffic classes present in the queues. Using a priority scheduling the queues are served in order of their priority. Isolation is the ability to isolate each flow from the others guaranteeing that they do not interfere with each other. Clark et al. described a schedule algorithm, WFQ [31], which was able to isolate the flows from each other providing a specific share of the available bandwidth to each flow. These techniques are the building blocks used in QoS approaches [36].

One of the solutions for supporting QoS requirements over the Internet was an integrated services model (IntServ) [29], with the goal of providing QoS to individual flows on the Internet. The model framework consists of four components: admission control, packet scheduling, classifier and reservation setup protocol. The idea was to have different service classes with different traffic characteristics that could match the application QoS requirements. IntServ establishes virtual paths for each flow and sets up the required resources in the path with the aid of RSVP [83], which is a reservation protocol that reserves a share of the resource for a service. According to [36] this solution was not scalable because the reservation mechanism was for each flow thus the router has to maintain the reservation for each flow that passes through it.

The differentiated services model (DiffServ) [26] was proposed to address the limitations of IntServ. Instead of providing QoS for each flow it provided for aggregates of flow by using 2 bits in the IP packet header to assign each flow to an aggregate behavior. Each aggregate received a different treatment in the network. DiffServ reserves resources for classes instead of flows thus removing the issue of scalability identified in the IntServ model. As the classification is based on the packet there is no need for a prior resource reservation thus eliminating the waiting time observed in the IntServ. According to [36] DiffServ does not provide end-to-end QoS assurances to the traffic in the Internet. Also implementing DiffServ can be difficult because it requires to define a large set of values for non specified parameters.

Recently there are new techniques that are being applied to improve the network availability and performance. [55] presented a protocol to detect faults in a bidirectional path between two forwarding engines, part of the router responsible for forward incoming packets. Their goal was to provide a short duration detection of failures in the path between two forward engines by periodically exchanging bidirectional forward detection (BFD) packets and if one stops receiving the packets over a certain period it assumes that the other node has failed. It was made to be used over any protocol layer and over any media. Another technique is the statefull switchover [77]. It allows a redundant engine to take over if the primary fails thus improving fault resistance. By maintaining the user session information during a switchover the backup engine is able to continue to forward the traffic thus improving the network availability.

One of the main issues with the existent routing protocols is that they converge only in seconds which is a lot of time considering the number of packet loss in that link. IP fast reroute techniques [73] can solve this problem by providing convergence in less than one second. One approach is to have pre-computed
backup paths so that when a failure is detected a backup path is used right away. The backup path has to be loop free by using a mechanism like split-horizon [81]. If is not possible to choose a loop free path it can be applied the NotVia mechanism which is more complex and requires new advertisement to be built. There are recent studies about using disjoint topologies technique [37, 20].

3.1.3 Overlay Networks

Overlays Networks are virtual networks built on top of other networks, for example a virtual network built on top of the Internet, as shown in Figure 3.1. Each node in the overlay network is a computer and they cooperate for the same purpose using application layer protocols. They have the ability to increase network resilience, which is the the ability to maintain a certain level of QoS in the network even in the presence of faults or changing conditions. This is done by allowing nodes to send packets to indirect paths instead of the direct ones provided by the underlying network protocols. By increasing the resilience the network is able to maintain the required QoS thus it is capable of providing timeliness assurances. There are several approaches regarding overlay network routing protocols but in this work only those that select paths in order to provide reliability assurances will be addressed.

Several overlay network routing algorithms choose the paths that provide better end to end performance metrics. However there is no overlay solution that aims to provide timeliness guarantees by selecting a suitable path accordingly to the deadline. Most of them aim to increase the resilience and the performance by quickly detecting a link failure or by using packet redundancy to mask network faults for example.

In [11] Akella et al. evaluated an overlay routing technique where each node had a complete view of the network thus they were able to select the best overlay paths in terms of performance and availability. They observed that this technique could improve round-trip time (RTT) between 20% to 70% compared
to using direct BGP [69] routes over a single internet service provider (ISP). Therefore it could offer better performance than the direct path provided by the Internet.

The authors of [18] described two approaches that took advantage of existing path redundancy: mesh routing based on packet replication and reactive routing based on adaptive path selection. In the mesh routing approach a node sends a message through multiple paths at the same time. The goal is that the message should arrive at the destination through one of the paths even if there are link failures. In the reactive routing approach the nodes periodically probe their links to determine the required network metrics allowing to select paths that have the best probability to deliver the message.

One example of the mesh routing approach was presented in [75]. The approach consisted in creating a mesh based overlay network where every node was connected to \( n \) other nodes, parent nodes, thus forming a high redundant overlay network. By having this type of network, a node could use multiple paths simultaneously to multicast a stream of data. The novelty in this solution was the use of a protocol which could reliably reconstruct a sequenced packet stream by receiving packets from multiple senders. By using this type of protocol, the receiver could detect that a packet was missing in the same way as TCP, by verifying the acknowledgment (ACK) numbers. When a certain time passes and it still did not receive the missing packet, the node would request the packet through all the senders it received from. This type of failure recovery has good results when there is a burst of packets to transmit because it can detect that a loss occurred faster. [75] evaluated this solution setting up an experimental procedure and concluded that a node can experience a loss rate of 5% even if the parents are experiencing a loss rate of 45% individually. The main issue of this solution is that the failure recovery is effective only when the detection time is very low, meaning it works fine for burst of packets but not for the control traffic type for example.

RON [17] is a reactive routing approach. It was described as an architecture that allowed nodes to detect and recover from path failures within seconds as compared to existent mechanisms in the underlying network that can recover only after a few minutes. In RON the nodes periodically probed their links and exchanged the measurements to other nodes in order to build a routing table which allowed to choose the best path available. By monitoring periodically each link a node could detect rapidly a link failure, consequently updating its routing table and then the node would disseminate this information to the others. RON used an exponential weighted moving average to calculate the expected latency in each link, similar to TCP. The loss probability is measured by calculating the two-way packet loss and then use the half for the one way packet loss probability because it is assumed that the paths are symmetrically. Assuming that paths are symmetric may not lead to the best results and it has been shown that Internet paths tend to be asymmetric [64, 45]. The authors of [17] also made some evaluations and they observed that RON took on average 18 seconds to detect and recover from faults. However this type of solution has a considerable issue which is that it is not scalable due to the aggressive probing and monitoring. Normally it works for a network comprised of 50 to 60 nodes. Although this solution presents a high resilience for the network it does not take into considerations the deadline of the messages thus it can not provide timeliness assurances.

There are other approaches using overlay networks to guarantee some level of QoS. The goal of
OverQoS [76] is to provide QoS over overlay networks by offering statistical loss and bandwidth guarantees between the nodes. It assumes that the path is predetermined and fixed. The solution provides these guarantees by first determining the level of quality of service that can be provided through that path and then uses a hybrid solution for error control combining FEC and ARQ to actually provide the assurances. FEC, forward error correction, is a mechanism that exploits redundancy in the information transmitted. FEC is mostly used when retransmissions are costly or nearly impossible. ARQ, automatic repeat query, is the opposite of FEC, as the receiver uses acknowledgments (ACKs) to notify that it received the message. In ARQ if the sender does not receive the ACK over some time, it retransmits automatically the message. This solution was built to improve QoS for streaming applications that work on top of overlay networks. This means that it achieves good results only for stream traffic type. In [16] it was described an architecture to improve the performance of VOIP applications built on top of overlay networks, denominated as High Quality VoIP Streams. As this kind of application depends on the latency and jitter provided, it was propose to use UDP. Therefore this solution tends to suffer from packet losses and network failures, as UDP is best effort. This solution involved a protocol that chooses the best path taking into considerations both the latency and loss rate in each overlay link, designated as expected latency cost function. The authors of [16] also presented a protocol responsible for recovering packets only if there was a chance of delivering it in time due to the application specification. This protocol tried to recover packets locally, meaning that the recovery could be done at each intermediary node instead of being done at the end-hosts. This protocol is similar to the one used in [75], but the main difference is that when the receiver detects that a packet is missing it triggers the retransmission procedure immediately. So the only variable for the failure recover procedure is the detection time. Because this protocol is more effective when the detection time is shorter, the nodes aggregate multiple voice streams and then forward them in one connection. Therefore they are able to reduce the detection time. One problem with this solution is that each node needs a lot of computational power to calculate the expected latency cost function of all the links of a path.

3.1.4 Multihoming

Multihoming can be defined as a computer connected to more than one external access link which belongs to different Internet providers or the same, as illustrated in Figure 3.2. The main advantage is obviously the increasing of the resilience in the network. Recently there have been studies about multihoming improving the network performance. In [10], the authors wanted to quantify the extent to which subscribers could leveraged connections to multiple network providers in order to improve the performance in a enterprise perspective and content provider perspective. The analysis was based on data collected from large data sets consisting of measurements taken at servers and measuring nodes of Akamai’s content distribution network (CDN). They concluded that choosing the right upstream provider was crucial to improve the performance. Choosing the wrong provider could lead to a performance twice as bad as the optimal choice.

The best approach for choosing the most suitable provider is to probe each upstream link in order
to know which can provide a better performance. Allowing the client to choose the best path to send messages and to which interface they should receive the messages was defined in [11] as *intelligent route control*. The authors of [12] evaluated several schemes that could extract the performance benefits of using multiple providers by offering routing control capabilities to clients. This work addressed how to monitor the links, how to choose the best path and how to direct the traffic over selected providers. One of the conclusion was that the route control schemes described could improve significantly the performance of client transfers up to 25%. Also they concluded that both active and passive measurement-based offered significant performance benefits in the range of 15% to 25% when compared to using the single best performing ISP provider.

However there are some considerations that have to be addressed in order to implement a multihoming solution. RFC 6418 [27] describes the main issues found when a node is attached to multiple provisioning domains or set of configuration information like gateways, DNS, prefixes and the correspondent interfaces. For each active interface the node can receive configuration information from different mechanisms such as DHCPv4, DHCPv6 or PPP. The problem is how to merge them or how to select the correct configuration when communicating. One example described in [27] was that when the configuration overlapped it could contribute to the node sending the data to the wrong destination. Other main issues are related to session management, routing, name resolution, address selection and security.

RFC 6824 extends TCP protocol to explore the multiples paths between peers, Multipath TCP [38]. The goal was to improve the throughput and resilience while also maintaining the same type of service that TCP offers. This solution operates at the transport layer and aims to be transparent to both higher layer and lower layer. To create multiple paths between two peers Multipath TCP establishes several TCP sessions, each associated to a different host address. Each TCP session is called a *subflow*. In the beginning the hosts exchange informations in order to know if they support Multipath TCP or not. This means this solution is backwards-compatible with TCP. This initial exchange also allows the hosts to establish additional subflows in the future. The send strategy used by the host is defined by a local
policy. Multipath TCP also provides data retransmission by sending it through the original and a backup subflow.

Such improvements provided by multihoming intelligent route control can help achieve the goal of providing timeliness and reliability assurances. However an issue is that the client can not control the entire path to the destination, he controls only the access links. This problem can be solved with overlay networks by choosing indirect paths to reach some destination as stated in the previous subsection. In [11], the authors made comparisons between overlays, multihoming and \( k \)-overlays which is a combination of multihoming and overlay networks in terms of RTT, throughput and availability. Between overlays and multihoming, they observed that in many cities as the number of providers increased, the RTT obtained by multihoming was shorter than overlays. The same applied to throughput as multihoming provided 2% to 12% better than overlays. When combining both overlay with multihoming, they observed that the benefits where marginally better than with just multihoming. However in terms of availability the overlay presented a 100% availability while multihoming not. In conclusion overlays combined with multihoming can offer better performance and resilience in the network than using just multihoming or overlays. Also with this combination is possible to provide a quick failure recovery.

One example of this type of approach, combination of multihoming with overlay networks, is MONET [19]. The goal was to improve client availability to web sites by one more nine or better through the combination of multihoming with cooperative overlay network of peer proxies. This degree of magnitude is related to the percentage of available time that service providers offer to clients. The authors of [19] described that normally this value is around 95% to 99%. This approach created many potential paths between clients and web sites and relied on a scalable algorithm to select a good path. MONET consists of a set of web proxies deployed across the Internet which serve as conduits for client connections to web sites. When a client sends a request, the MONET proxy first checks if it has the object in its cache. If not it will check if it already has an open connection to the web site and if so it uses it to obtain the object. Otherwise it will use the waypoint selection algorithm to obtain an ordered list of the paths to the site and connects to the servers in the specified order. Each element of this list has a delay which specifies the time to wait until the correspondent path should be probed again. This algorithm seeks to order these paths accordingly to their degree of success. Occasionally MONET uses paths that are not the best to check if their quality has changed. In order to know when to do a retransmission in case of failure, MONET uses an algorithm to adapt the delay between requests depending on the network conditions. Although this algorithm does not have an high precision it is lightweight and can be implemented without increasing the computational requirements. The authors concluded that the waypoint selection algorithm performed almost as well as sending requests to all available interfaces, flooding. Although this system achieves most of the requirements stated for this work, its goal is to improve the resilience and performance meaning that it does not take in to considerations the message deadline in order to select a good path. Also it can not offer a good reliability in terms of delivering message in time because the algorithm used to adapt the timeout is not very accurate.
3.1.5 Data Center Networks

Guaranteeing timeliness assumptions is an important objective in today's data centers. It is a big issue because they tend to host multiple applications and displaying latency to the end users promotes a bad user experience, which leads to loss of revenue. The problem is that the hosted applications tend to generate different types of flows at the same time. Therefore, it becomes difficult to manage all those flows without compromising packet loss and consequently missing deadlines. One of the solutions is to over-provision the network bandwidth by providing far more bandwidth than the necessary. Another approach is to implement a deadline-aware network that allocates the required bandwidth for each flow, if available.

Wilson et al. proposed a solution to maximize the application throughput by maximizing the number of flows that are completed within the deadline [82]. Assuming that at the flow time initiation the corresponding size and deadline were available, the protocol tried to allocate in each router along the path the needed bandwidth in a greedy approach. This means that each router would try to satisfy as many flows as possible and the rest would be given a minimum bandwidth. Also in [82], they evaluated this solution using a small testbed. The conclusion was that their solution supported 28 concurrent senders while assuring that at least 99% of the flows completed in time in a moderate deadline, around 30 ms. Although this solution achieves good results in terms of providing timeliness assurances, it mandates that the network elements, like switches, have to be changed in order to support the allocation. Also it does not take the value of the deadlines in the bandwidth allocation procedure. In fact [78] described that this solution allocated the bandwidth for some of requests with longer deadlines before the ones that had smaller deadlines.

D2TCP is a distributed and reactive deadline-aware protocol which provides timeliness assurances and also alleviates the network congestion [78]. This protocol adds deadline awareness while also providing TCP properties like the reactivity for allowing end-hosts to correct the over-subscription of the network. This solution also avoids network congestion issues by using an algorithm to prioritize the flows accordingly to their deadlines. To evaluate the protocol the authors did a small scale real test and simulations for large scale, and they concluded that the protocol reduced missed deadlines compared to D3 [82] by 50%. This protocol can co-exist with existent transport protocols like TCP.

Another type of approach to provide timeliness assurances was described in [52]. The authors adopted a software defined network (SDN) architecture for the interconnection of their data centers. A SDN allows the separation of the control plane from the data plane for the network management thus simplifying networking to enable rapid deployment of new network control services. The timeliness assurances can be provided by the centralized traffic engineering service because it is responsible for the allocation of the bandwidth among services basing on their priority. The authors of [52] observed that this solution enabled them to deploy substantial cost-effective WAN bandwidth, running many links at near 100% for extend periods thus providing high resilience in the network.
3.1.6 Discussion

In conclusion, none of the approaches described in this section are able to guarantee the timely and reliability assurances in the delivery of messages over the Internet while meeting the proposed requirements.

Table 3.1 compares some of the solutions described in this section according to the requirements that were stated earlier because none of them were evaluated with the same benchmark. The solutions are compared in terms of: providing timeliness assurances, providing resilience over the network and providing a quick failure recovery. The last term means that if they can detect quickly if a failure occurred and they are able to recover from it.

As the reader can see, the data centers approaches are able to provide timeliness assurances. However they rely on having a dedicated underlying network that is able to provide high available bandwidth which is not the case for this work. Normally building these type of network have a high cost which only big companies are able to afford it, for example Google which uses B4 [52] to connect their data centers.

ATM [25, 33] could provide timeliness assurances by allowing the allocation of fixed cells on demand. ATM defines four type of classes for QoS and each one establishes the required metrics. Also it provides service categories. These categories allow the users to select specific combinations of network parameters that are adequate to the intended service. However due to its complexity, ATM is not highly used today by end-host applications.

Multimedia applications have some of the requirements proposed for this work, mainly the timeliness assurances. To provide these assurances normally they rely on mechanisms like Bidirectional Forward Detection [55] and statefull switchover [77] to achieve it. But these mechanism are not reliable in the delivering within deadline because they are not able to quickly converge in case of a failure in the network. Also they do not adapt to the conditions that the underlying network is providing. DiffServ [26] is an interesting solution but it still has some issues to resolve as described in [36]. For example as DiffServ does not reserve the path therefore packets could be lost in case of network congestion. There are new works that intend to reduce the convergence time by having pre-computed backup paths such as disjoint topologies technique [37, 20].

Multihoming and overlay routing approaches can only provide resilience in the network and quick failure recovery by relying on mechanisms to detect a failure and select a backup path quickly. The issue with multihoming approaches is that although they add redundancy in the paths it is not possible to control the entire path to reach a destination. The client only chooses the access link and the rest is chosen by the by the network-level routing protocols, namely BGP. The only way to control the entire path is to modify BGP, more precisely changing the path selection algorithm to take into account other network parameters such as latency or packet loss rate. However this is a difficult task because the network providers normally have agreements between them about the traffic exchanged. Overlay routing can solve this issue by providing the client an indirect path to reach the destination, but it does not add the same degree of redundancy as multihoming.

Therefore none of the existent solution can provide all of the requirements proposed in this work, namely none of them is deadline aware for network control traffic and for file transfer. Nevertheless there
are some good lessons can be taken from these works as most of the presented solutions improved the resilience or the performance over WANs.

### Table 3.1: Comparison of the existent solutions according to the proposed requirements

<table>
<thead>
<tr>
<th>Solution</th>
<th>Approach</th>
<th>Timeliness</th>
<th>Resilience</th>
<th>Quick Failure Recovery</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>QoS/ATM</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>[61]</td>
</tr>
<tr>
<td>DiffServ</td>
<td>Resource</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>[26]</td>
</tr>
<tr>
<td>Allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTCP</td>
<td>Multihoming</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>[38]</td>
</tr>
<tr>
<td>MONET</td>
<td>Multihoming</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>[19]</td>
</tr>
<tr>
<td>Mesh XML</td>
<td>Overlay</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>[75]</td>
</tr>
<tr>
<td>RON</td>
<td>Overlay</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>[17]</td>
</tr>
<tr>
<td>D2TCP</td>
<td>Data Center</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>[78]</td>
</tr>
</tbody>
</table>

#### 3.2 Network Parameters Estimation

In order to assure some level of QoS to the clients, most ISPs monitor the network by measuring the parameters like latency, bandwidth, jitter or packet loss in a consistent basis. By knowing what is the current condition of the network they apply some techniques, for example the ones discussed in Section 3.1.2 in order to provide the required QoS to the clients. Therefore it is important to estimate those parameters with high precision and low overhead. This section is going to address the network parameters and the tools to estimate them that are relevant for this work. Only tools that measure these metrics on an end-to-end perspective will be addressed.

##### 3.2.1 Round-Trip Delay

Network delay is an important indicator of the current state of the underlying network. It shows the time that a packet spends in the path until it arrives at the destination node. One type of network delay is the round-trip delay which can be defined as the time spent since the sender sent a request until receiving the correspondent response. According to [15], the round-trip delay is useful for applications that do not perform well if the delay is above some threshold. Also values above the minimum delay provide an indication of the congestion present in the path. There are two ways of measuring round-trip delay: active measurement and passive measurement. In active measurement a host is periodically probing the network while in passive measurement the host relies on data to be transmitted in order to obtain the round-trip delay. Therefore the passive approach is only effective if it is generated a substantial amount of traffic.

There are several approaches to estimate this metric. They differ in the types of packets that are used and the protocol layers where the measurements are made. One of the most popular is the ping tool which takes advantage of the request and reply mechanism defined in the ICMP [67]. It estimates the delay by measuring the time that the exchange of echo request and echo reply packets take. It is very popular because it is already included in the IP and is available for a long time in the majority of the operating system. However there is a major problem with this approach. According to [66] it was
observed some unexpected measurements using ping. One explanation was that the load-balancing mechanism implemented by routers were affecting the flow that the packets would take. This mechanism were implemented by routers that supported ECMP [48] by applying hash functions over the packet's header. Another approach is to use the TCP three-way handshake by calculating the time between the SYN and SYN/ACK packets.

One different approach to measure the round-trip delay between two hosts is measuring the RTT between their authoritative name servers. This approach was proposed in [41], with the goal of creating a tool that is scalable and does not need active cooperation from the two end-hosts. It assumes that most of the end-hosts in the Internet are close to their respective authoritative name servers. The authors gathered data from a list of servers previously studied and observed that 76% of web servers and 79% of Napster clients had at least one authoritative name server that was close to them.

Nevertheless all of these approaches have one thing in common which is that they use timestamps to mark when the packet was sent and when it was received. [15] discussed that one issue of measuring round-trip delay is that the clock should be synchronized since the request was sent until the response arrives. During this time if some event discontinued the clock it could lead to an inaccurate estimation. Also the authors discussed that using software clocks could cause some level of uncertainties because they only marked the times in the application level. Therefore they proposed that it should be used wire times, time in the network interface.

### 3.2.2 One-Way Delay

In todays Internet the response path may be different from the request path, thus measuring the RTT does not provide the accuracy that some applications need as they are more concerned about the forward delay. Most applications calculate the one-way delay (OWD) by taking the half of the measured RTT. This is equivalent to assume that the delay is symmetric, meaning that both forward and backward delay have the same value. [64] wanted to quantify the delay asymmetry because of the path asymmetry existent on the Internet. After analyzing several measurements made by owping [5] which is a one-way delay estimation tool, and comparing them to the correspondent RTT, they observed that delay asymmetries definitely exist in the Internet. For example the RTT of a measurement was 150 ms and the respective forward delay was only 60 ms. Also they concluded that delay asymmetry implies that there is a path asymmetry but not vice-versa.

To the best of our knowledge all approaches for measuring OWD accurately require synchronized clocks. However synchronizing the clocks is not an easy task. RFC 2679 discusses some challenges to face related to clock synchronization like clock uncertainty, precision and skew, when calculating OWD [14].

There are two approaches to synchronize clocks in order to calculate the OWD. The first one involves that the hosts have to synchronize their clocks directly, accessing each other or using an external node. One example is the owping. Before sending the packet the host node marks its current time on the packet. When the destiny receives the probe packet it subtracts its current time with the time on the
packet thus obtaining the OWD. For each measurement the hosts exchange 10 probe packets. Owpings uses NTP [59] to synchronize the hosts clocks. Before starting any measurement owpings obtains the clock drift, the difference between host's and NTP's clock, from the NTP daemon on the system.

The other approach is by deploying proper infrastructures in the nodes which run a dedicated software. One example is Surveyour [53], which deploys measurement infrastructures at the sites and these infrastructures all have their clock synchronized. Each infrastructure performs its own measurements and stores in a central data base. Surveyour uses GPS for clock synchronization. GPS is an alternative for NTP but it requires a proper hardware.

### 3.2.3 Available Bandwidth

Estimating accurately the available bandwidth of a path can benefit applications that require this parameter, like file transfer or interactive. It benefits because the applications can choose the paths that can support the size of their packets that they are about to send. As this work also pretends to assure timeliness in file transfers, measuring the available bandwidth in each path is important. [68] defined the available bandwidth as the unused or spare capacity of the link during a certain time period.

Iperf [2] is the easiest and most used tool. Its goal is to measure the achievable throughput using parallel TCP connections using TCP or UDP data streams. According to [50] the available bandwidth in a path could be estimated by measuring the achievable throughput.

Nevertheless there are other approaches. The authors of [68] described that there are two main techniques to estimate the available bandwidth: self-loading periodic streams (SLoPS) and train of packet pairs (TOPP).

In the SLoPS technique the sender sends \( k \) number of packets with the same size to a receiver in a rate \( r \). It relies on the measurement of the one way delay of theses packets to verify if the delay is increasing or not, and if so it means that the rate is bigger than the available bandwidth in the path. The goal is to bring the rate close to the available bandwidth using an iterative algorithm. One example that uses this technique is Pathload [51].

In the TOPP technique, the sender sends each time a pair of packet probes to the receiver with an initial delay between them. In the receiver if the packets do not arrive with the same delay between them, TOPP assumes that the send rate is higher than the available bandwidth. The difference between these two techniques is that TOPP increases the rate linearly to reach the available bandwidth. One example that uses this technique is ABwE [63].

[74] made comparisons between these tools in terms of accuracy and overhead introduced. They observed that ABwE was the least accurate however it introduced the lowest overhead, almost none. Between Iperf and pathload, the authors observed that pathload was more accurate and introduced less overhead than Iperf. This analysis was made using data with realistic workload characteristics and also they used generated traffic from their lab, and they compared the estimated values with the values obtained by MRTG which is a network monitoring tool.
3.2.4 Packet Loss

Packet loss can impact application’s performance whether they use any transport layer protocol by degrading their quality of service. For example when using UDP, excessive packet loss will impact negatively the user experience in applications like VOIP or video conferencing.

There are several tools that estimate the packet loss over a path. Ping and Iperf are two of the most used due to their simplicity. Ping uses ICMP packets as probes and it sends them at a fixed time interval. The loss is detected when the ICMP echo response does not arrive in a certain time period. Iperf uses the same methodology but the difference is that it uses UDP packets instead of ICMP packets.

One interesting finding is that packet loss also presents asymmetries over a certain path. In [28], it was observed that like the delay, the forward loss rate was not equal to the backward loss rate mainly due to the path asymmetry that the Internet provides. Sting [72] is a one-way packet loss rate estimation tool. It explores the TCP’s error control mechanism to estimate the packet loss rate in both forward and reverse path. However Sting requires changes in the TCP so that it could provide accurate estimations.

Another interesting property of packet loss is that it presents some dependence pattern. In fact [28] concluded that the probability of a packet to be lost in a path was higher if the previous one was also lost.

3.2.5 Path Diversity

The diversity of the multiple paths between two end-hosts impacts the performance and the availability in the network. For example [43] observed that although multihoming or overlay networks could provide significant gains in the performance and availability, their effectiveness depended of the natural diversity of the redundant paths between the end-hosts and in the ability to select the most uncorrelated paths. They concluded that to improve these metrics the overlay networks or multihoming should be topology-aware.

There are several solutions to estimate the correlation between the multiple paths available. The easiest approaches rely on using tools like traceroute [57] to determine the collection of routers that are present in a particular path. The same result can be accomplished using the using lft tool [4]. The difference between them is that lft determines the collection of autonomous systems (ASs) present in the path instead of the IP addresses. In this type of approach the correlation between the paths can be calculated by using the number of the ASs or IP sub-networks that are present in both paths. Most applications use these approaches because it is easy to deploy and to compute the correlation value. Traceroute is widely used because it is already implemented in IP. Paris-traceroute [21] is an interesting solution because it can handle anomalies present in the network that traceroute cannot. The authors of this work illustrated that the anomalies were caused by load balancing mechanisms present in the routers. Paris-traceroute is able to handle these anomalies by controlling the content of the packet’s header.

However there are other approaches which use complex computations to determine the correlation between two paths also adding new metrics like the path history or link failures. One example is the
solution presented in [84], which exploits the paths availability history in order to select topological dis-
joint paths. They proposed a new mechanism called availability history window (AHW) which records 
the failures that occurred in the path over a time interval and from which the correlation of two paths can 
be determined. From the AHW they can determine that two path are highly correlated if they tend to fail 
at the same time interval. With this type of scheme they were able to detect and turn away from failure 
correlated paths.

In [1] the authors tried to find a backup route that minimizes the joint path failure probability between 
the backup and the primary path. They proposed this scheme because the overlay paths that share 
the same physical link would present a highly correlated failure probability. Therefore they are able to 
select a backup path which share the least failure probability in the overlay path but also sharing the 
least physical link in the network.

3.3 Dependability in Message Delivery

Achieving a defined level of dependability in the delivery of the messages over the Internet is a hard 
challenge due to the unpredictable environment which the Internet is. One way of providing dependability 
is to have a way to adapt the time bounds, timeout values that triggers the retransmission process, 
accordingly to the environment provided by the underlying network. Another solution tends to be by 
proactively monitoring the network channels in order to know in advance if some channels may fail or 
not. By resorting to these type of approaches a system can behave correctly even if the environment is 
uncertain.

3.3.1 Dependable Adaptation in Uncertain Environments

In order to provide timeliness assurances systems have to adapt to the conditions that the underlying 
network is providing in order to preserve the required QoS. One way to provide dependability in the 
delivery is to timely detect failures, in this case timing failures, in order to trigger the retransmission mechanism while there is still time to deliver the message. This means that the failure detector has to 
be reliable in such way to detect the failures in time and also to be sure that the failure really occurred.

Most of the solutions presented in Section 3.1 have a mechanism to rapidly detect timing failures. 
For example in [16] and [75] it was detected that a failure occurred when the receiver did not receive the 
packet $n$ while receiving packet $(n + 1)$. This means that if the receiver does not receive the following 
packet of the sequence it does not detect the failure, thus this mechanism is efficient only when there is 
a stream of packets going through the network.

[19] described two mechanisms for triggering the retransmission protocol, $rttvar$ and $k$-means clus-
tering. The approach of these mechanisms is to set up a delay threshold between the requests that 
should adapt to changes in the network conditions. The k-means clustering scheme collects connect 
time samples and groups them into k clusters. This computation can be done periodically or each time 
that a connection failed or succeeded. The $rttvar$ scheme is inspired on the TCP retransmission timer
computation described in [65]. Basically for each RTT sample taken, the delay threshold for the next requests will be updated computing the average linear deviation of the delay until the path is probed again. It is simpler to compute the delay threshold using this scheme than using k-means clustering but the downside is that is less accurate than k-means clustering.

The main challenge to provide this type of dependability is how to adapt the delay threshold to changes in the network conditions without compromising the failure detector accuracy.

The authors of [79] presented the *Timely Computing Base* (TCB) which addresses this problem. In this work, the authors explored the concept of coverage which is the degree of correspondence between the system timeliness assumptions and what the environment can guarantee. Also they defined the coverage of a timely property as the probability of that property holding over an interval of reference. Basically the idea is to compute a time bound that assures that a timely property is guaranteed with some probability $p$. By using these concepts, the authors were able to compute the probability distribution function, pdf, which represents the distribution of the timing variable with a known error, $p_{dev}$. They defined that an application whose bounds, delay threshold, can be changed dynamically belongs to the *time-elastic* class which is one of the classes that TCB can support. The authors of [30] applied these concepts to build a dependable QoS adaptation mechanism. The idea is to generate several pairs of (bound, coverage) to specify the QoS requirements, meaning that each time bound holds with certain coverage. By using these pairs, the system can select the new bound when the network conditions change in order to maintain the same coverage.

This means that *dependable adaptation* can be defined as ensuring that effective coverage of a system property will stay close to the assumed coverage and the difference is bounded.

The authors of [35] improved the approach in order to maintain correctness of the system properties after adaptation. With this improvement, the system was able to select optimistic time bounds when the environment was stable and select a conservative bound when the environment was unstable, but still providing dependable adaptation. The process to calculate an adequate time bound is divided in two main activities. First the system identifies whether the network is stable or not. This is done selecting a distribution model that characterizes better the obtained samples. The distribution models considered in this case are the Conservative, Exponential, Shifted Exponential, Pareto, Weibull and Uniform models. The time bound value will be calculated accordingly to the selected distribution model. If none of these models can characterize the samples then it is considered that the network is unstable. In this case it will be selected a conservative time bound.

In conclusion, the objective of dependable adaptation is to select the adequate time bound, so that the effective coverage of a property will be bounded to some specific value [35].

### 3.3.2 Network Channel State Prediction

Systems that proactively handle failures can be more dependable because if they know a situation or event in advance they can trigger countermeasures in order to prevent the failure to occur or to repair it. [80] describes two types of prediction related to the time horizon, short-term prediction and long-term prediction.
prediction of events. The short-term prediction of events is easier and more successful than the long-
term prediction. With short-term prediction systems can prevent or limit the damage that can be caused
by a near future failure. [71] describes that proactive fault management consists of four steps:

- 1. The online failure prediction which collects a sample of measured data and outputs a decision;
- 2. The diagnosis phase, which is triggered by the online failure prediction to find the error;
- 3. The action scheduling which takes the decision and selects the actions that should be used for
countermeasure;
- 4. The execution of those actions.

Thus in order for proactive management to be efficient, the online failure prediction model should be
accurate.

Online failure prediction models are based on the runtime monitoring of the system parameters
during a small interval to predict whether a failure might occur or not in the near future. Figure 3.3
illustrates how the online failure prediction works. At time $t$ it is predicted that a failure might occur from
$\Delta t_1$, the lead time. The failure is predicted by analyzing the measured data during $\Delta t_d$. The prediction
is valid for the interval $\Delta t_p$, designated as prediction period. The interval $\Delta t_w$ is the warning time which
is the time that the system needs to react or prepare for the upcoming failure. This time needs to be
smaller than the lead time in order to prevent or countermeasure correctly the future failure that was
predicted.

![Figure 3.3: Online Failure Prediction definition](image)

There are several approaches and algorithms for online failure prediction. [56] described some
metrics to qualify and compare the approaches. These metrics are mostly used by information retrieval
algorithms but they can be applied to online failure prediction algorithms. To compute these metrics
the authors of [71] based on the contingency table, which described the concepts of true positive, false
positive, true negative and false negative. True positive (TP) indicates that the algorithm predicted a
failure and it occurred within the prediction period. False Positive (FP) indicates that the algorithm
predicted a failure but it did not occurred within the prediction period. True negative (TN) expresses that
the algorithm did not predict a failure and no failure occurred within the prediction period. False negative
(FN) expresses that the algorithm failed to predict a true failure. The most relevant metrics are precision,
recall, F-measure and accuracy.

Precision is the proportion of correctly predicted failures against all the failure predictions made:
\[
\text{precision} = \frac{TP}{TP + FP} \quad (3.1)
\]

Recall is the proportion of the correctly predicted failures against all the failures that occurred:

\[
\text{recall} = \frac{TP}{TP + FN} \quad (3.2)
\]

Precision and recall measure the ability of a system to correctly predict the failures. However by improving one the other decreases. Another metric is the F-measure which is the harmonic mean of precision and recall:

\[
\text{F-measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (3.3)
\]

Accuracy is the ratio of all correct predictions made to the number of all predictions that were made and not made:

\[
\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \quad (3.4)
\]

The authors of [71] proposed to divide the approaches into five major branches. These branches differ in the type of input data that is measured. This section is going to address only two of those branches: symptoms and detected errors approaches. The other branches are not addressed because they can not be related to this work.

Symptom based prediction addresses the manifestations of faults that do not cause failures right away. Normally they are referred as side-effects of an error. The main challenge here is the symptoms are hard to detect. One example is a memory leak in the system because it does not cause any error until there is no more memory available. Nevertheless a memory leak can be detected by monitoring the memory usage and if it shows an increasing value it may be due to leaks. This concept can be applied to network by relating symptoms to the available bandwidth in the path, because if this parameter is constantly decreasing it may be a consequence of network congestion in the links of the path. The same thing can be applied to the path latency. There are four types of approaches in this branch which are: function approximation, classifiers, system models and time series analysis.

The goal of function approximation approaches is to select a function that closely matches a target value given an input data. The output of this function can be the probability of a failure occurrence or the future values of computing resources. One example of this approach is linear regression, which adapts the parameters of a linear function in order to the resulting curve of the function matches the measured data. This approach can be used to predict for example the number of calls in a telecommunication operator or its availability. By applying this type of technique, the authors of [46] were able to model and forecast the call availability and failures of an industrial telecommunication system.

Classifiers can also be used to predict failures. Using classifiers, the system classifies the current state as a failure prone or not state. The classifier has to be trained first by collecting samples of failure prone state data and correct state data. Then it computes a graph which has a point that separates
the failure prone states from the correct states. In runtime, given the current state the classifier checks whether this state is on the part of the graph which correspond to the failure prone state or to the correct state. In order to improve its accuracy, it is need a large amount of data for the training phase.

System models are similar to classifiers but the difference is that for the training phase they only need the data that corresponds to the correct state. During the runtime if the current state does not match the correct state, it is assumed that is a failure prone state. Although this method requires less data to be trained it has a higher false positive rate than classifiers. The authors of [47] proposed an intelligent system for failures detection that learns the normal behavior of the network and any deviation from this behavior are detected. The information is gathered and combined in the probabilistic framework of a Bayes network. The main advantage of this solution is that it could detect unknown faults. Also this type of approach is used in Network Intrusion Detectors in order to detect recent types of intrusions.

The last approach is based on time series analysis which treats a sequence of measured data as time series. This type of approach is mostly used to predict the future progression of the series in order to estimate the future value of the parameter. One example of its applicability is to compute the time for the resource exhaustion. Also it can be used to predict if the time series will violate a certain threshold.

Detected error reporting based prediction uses error reports to predict an upcoming failure. Basically it analyses all the errors that occurred during a time interval to predict a failure. The main difference in relation to symptoms is that the input is event-based instead of periodically monitoring the parameters. According to [71], this branch can be divided into 5 approaches. One of them is based on classifiers which have been discussed above. The other ones are: Rule-Based Systems, Co-occurrence, Pattern Recognition and Statistical Tests. Rule-Based Systems predicts an upcoming failure if at least one of the defined conditions are met. The conditions or rules are defined through error reports. Co-occurrence predicts that detected errors that occurred together are likely to occur again. Pattern recognition computes a ranking value to a sequence of errors reported based on the similarity to patterns that are prone failures and patterns that represent the correct state. Then through a classification based on the rankings, it is made a prediction. Finally Statistical Tests, like the name says, applies statistical tests to the reported errors to predict an upcoming failure.

These approaches rely on a training data set of samples in order to train the algorithms in order to detect failures in run time. [56] describes them as supervised machine learning techniques, because they involve a training phase.

Most of the works in this area use these techniques as base. For instance the authors of [46] used a non-linear regression model to forecast call availabilities and failures of a telecommunication system. In [70], the authors used a model based on the recognition of failure-prone patterns to also predict failures in a telecommunication system. Another example is the in the work [13] where the authors presented an finite-impulse-response (FIR) model to predict internet network traffic volume.

Another approach to predict events is through the analysis of a time-series event data base to obtain periodic episodes, i.e., sequence of events that occur frequently. This means that if we are able to find periodical episodes the probability of them occurring again is very high.
For example the authors of [58] applied multiple algorithms to discover episodes in a telecommunication system alarm log. Those algorithms were based on the idea of finding less frequent events and then proceed to find episodes with larger frequency period.

Also there are some works that aim to discover episodes with only partial periodicity. The distinction of full periodic with partial periodicity is that in full periodicity every point in time has to contribute to the frequency of the episode while in partial periodicity only a subset has to contribute. In other words, an episode can be considered partially periodical even if it does not occur in all of the points in time. For instance the authors of [44] presented multiple algorithms to find partial periodic episodes by exploring partial periodicity properties like for example the Apriori property where an episode is defined as periodic if it has a minimum number of occurrence.

These kind of approaches can be applied in a network context to parameters like those discussion in Sections 3.2.1-3.2.4. They can be used to:

- Predict which network channel is in a dangerous state and may fail in the future;
- Predict the future values of the latency, available bandwidth and packet loss;
- Predict which routers will be used on a path in the future.

3.4 Summary

Realizing that there is no existent solution that is able to provide timeliness and reliability assurances while also providing dependable behavior, this work pretends to fulfill those requirements. The first step is to monitor the network parameters. Section 3.2.2 described the parameters that will be measured and the correspondent approaches. Choosing the best approach is important because it could impact negatively the network and in the worst case causing failures.

Table 3.2 compares some of the approaches used for estimating the delay in a path indicating: their target, type of network delay, the approach used, and presenting their main advantages and disadvantages. OWD offers more advantages mainly because it is possible to estimate the forward delay on a path which is very important for applications like VOIP. However is very challenging to estimate the OWD accurately because the clocks have to be synchronized. Between the two approaches described, estimating the OWD by accessing the hosts is less expensive than using a dedicated infrastructure but it needs that both hosts clocks are synchronized meaning that it is an intrusive approach. The most used approach is through using ICMP packets, as ping does, but as explained in Section 3.2.1 it is not very accurate. King [53] is an interesting approach because it does not need direct interaction between the hosts, but it relies that the DNS servers used are located near the hosts.

Another network parameter described was the available bandwidth on path. There are three types of approach and in this case all of them were tested using the same benchmark which helps the comparison. Table 3.3 compares the three approaches, presenting one example for each of them. Iperf is very accurate but it is sensible to packet loss meaning that it can loose the accuracy if it occurs packet loss
Table 3.2: Comparison between Delay estimation tools.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Target</th>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>RTT</td>
<td>Host interaction</td>
<td>Implicit in IP</td>
<td>Accuracy</td>
<td>[6]</td>
</tr>
<tr>
<td>King</td>
<td>RTT</td>
<td>DNS interaction</td>
<td>Accuracy</td>
<td>DNS located near the host</td>
<td>[41]</td>
</tr>
<tr>
<td>OwPing</td>
<td>OWD</td>
<td>Host interaction</td>
<td>Forward Delay</td>
<td>Clock Synchronization</td>
<td>[5]</td>
</tr>
<tr>
<td>Surveyour</td>
<td>OWD</td>
<td>Infrastructure</td>
<td>Forward Delay</td>
<td>Dedicated Infrastructure</td>
<td>[53]</td>
</tr>
</tbody>
</table>

During the estimation, Pathload [51] is an interesting solution but its accuracy depends on estimating the OWD accurately. ABwE [63] provides less overhead for the network but it is not accurate.

Table 3.3: Comparison between Available Bandwidth estimation tools.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iperf</td>
<td>Parallel TCP</td>
<td>High Accuracy</td>
<td>Sensible to Packet Loss</td>
<td>[2]</td>
</tr>
<tr>
<td>Pathload</td>
<td>SLOPS</td>
<td>High Accuracy</td>
<td>Relies on OWD</td>
<td>[51]</td>
</tr>
<tr>
<td>ABwE</td>
<td>TPP</td>
<td>Less Overhead</td>
<td>Low Accuracy</td>
<td>[63]</td>
</tr>
</tbody>
</table>

Packet Loss estimation was also described. Table 3.4 compares the approaches previously described in terms of their advantages and disadvantages. The easiest way of estimating it accurately is using ICMP packets because they are already deployed in the IP. Therefore the majority of systems use it. However they can be easily filtered by firewalls. Iperf is easy to use but adds more overhead on the network. Nevertheless there are some more complex approaches that take into consideration the path asymmetry problem because like for delay, the degree of packet loss can be different for each way in a path. Sting is one example of this approach. It estimates both reverse and forward packet loss rate by exploring the TCP’s error control mechanism. However the TCP has to be modified in order to obtain accurate results.

Table 3.4: Comparison between Packet Loss Rate estimation tools.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>ICMP</td>
<td>Implicit in IP</td>
<td>Can be filtered</td>
<td>[6]</td>
</tr>
<tr>
<td>Iperf</td>
<td>UDP</td>
<td>Easy to use</td>
<td>High Overhead</td>
<td>[2]</td>
</tr>
<tr>
<td>Sting</td>
<td>One-Way</td>
<td>Forward Packet</td>
<td>TCP modification</td>
<td>[72]</td>
</tr>
</tbody>
</table>

Estimating the Path Diversity is important for systems that rely on retransmitting messages over a backup path, because they want to choose paths that are not correlated. Table 3.5 compares the approaches indicating their respective advantages and disadvantages. Traceroute is widely used because it is already implemented in IP. Lft is an alternative for traceroute. These two are different because they are IP aware and AS aware, meaning that it searches for IP addresses and ASs over that path respectively. Traceroute is less accurate than the other solution because a single router has multiple IP addresses. Lft provides low granularity because there are AS that have a large number of routers. Therefore it may be possible to discover just one AS in path. There are some other complex approaches like the one presented in [84] which correlates paths using their history of link failures. These approaches are more accurate than just using IP addresses or AS providers discovered in the paths to correlate
them. However they rely that the failures have to be captured accurately in order to provide accurate results.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traceroute</td>
<td>IP addresses</td>
<td>Implicit in IP</td>
<td>Low Accuracy</td>
<td>[57]</td>
</tr>
<tr>
<td>Lift</td>
<td>AS</td>
<td>Easy to use</td>
<td>Low Granularity</td>
<td>[4]</td>
</tr>
<tr>
<td>AHW</td>
<td>Path History</td>
<td>High Accuracy</td>
<td>Clock synchronization</td>
<td>[84]</td>
</tr>
</tbody>
</table>

Finally the last requirement for this work is that it should provide a correct behavior even if the environment is changing over the time. One way is to dependably adapt the timeouts for the timing failures accordingly to the network state. Most applications use a fixed value thus they do not take advantage of what the underlying network is providing or when the conditions degrade they loose the dependable behavior. The authors of [30] described a dependable model that adapts to the environment which relies on the concepts of coverage and built a probability density function that describes the probability distribution of the current state. By using this function it is possible to know which bound should be used to trigger the retransmission depending on the coverage needed. Another way is to proactively handle failures by predicting when they will happen in the near future. To make this possible, techniques like classifiers, system models, time series analysis, function approximation and so on are used. These techniques differ on the inputs used and on the type of prediction they provide. Classifiers and System Models are best used to predict the state of a target. They can be used to know the opinion of a sentence for example. Time series analysis and function approximation are best used to predict a value in the near future. The accuracy of these techniques relies on the amount of data that are used in the training phase. This phase is important because it allows the tuning of the algorithms thus improving the accuracy. Another approach is through mining of events stored in a data base. The goal is to extract periodical episodes, sequence of events that occur frequently, which can allow systems to predict that the same episodes may have an high probability of occurring again in the future in the same period.
Chapter 4

TimelyWare

As seen in the previous chapter, none of the existent solutions can assure all of the intended requirements for this work. Therefore we propose a new solution that is able to meet those requirements, TimelyWare. In this chapter the TimelyWare architecture and the implementation details of each component of the system will be presented. Specifically about the implementation it will be illustrated which tools were used to help the construction of the prototype as why they were chosen.

TimelyWare is an evolution of an algorithm called Calm-Paranoid (CP) [32]. That algorithm aims to deliver short (control) messages on time. The design rationale of CP was driven by the architecture of modern wide-area critical information infrastructures such as the power grid, which are distributed over several facilities following a WAN-of-LANs model. CP also uses overlay networks combined with multihoming where each node is connected to all of the other ones, following a one-hop source routing scheme [17]. CP periodically monitors the network in order to assess the degree of independence between the channels and to measure the network latency of each one. When a node wants to send a message, it selects the base channel and $B$ backup channels based on their delay and correlation to the base channel. The value of $B$ is predefined and constant. The concept of base channel is the channel that allows the most retransmissions attempts but can still delivers the message within the deadline. This means that its latency has to be less than the intended deadline.

Unlike CP, TimelyWare tries to adapt to the current environment by periodically estimating the retransmission time for each channel accordingly to their history and by using mechanisms to predict their future state. These mechanisms allow increasing the timely delivery probability and reducing the number of transmitted messages. Another difference is that in TimelyWare the number of backup channels is calculated dynamically depending on the remaining deadline and the state of the environment.

4.1 Architecture

We consider a system composed of several nodes connected by an IP network as illustrated in Figure 4.1. Each node is connected to all of the other nodes that comprise the system which enables the existence of more than one way to reach all of the nodes, i.e., network redundancy. This type of archi-
Architecture allows creating one-hop routing overlay channels, i.e., logical connections between two nodes that can be used to reach a certain destination node over different network-layer routes (because the intermediate nodes are different).

The goal of this architecture is that critical infrastructure facilities or data centers nodes are able to connect to the TimelyWare nodes using a point-to-point connection and use them as a form to deliver their messages. However, there can be TimelyWare nodes that are not connected to any facility nodes and still be part of the network. These particular nodes serve as gateways that can relay other node messages. This way, if there are few facilities, it is possible to have a large number of channels by deploying these relay nodes in the network. For example, if we have a network with two distributed facilities, instead of having just one logical connection between two TimelyWare nodes, we can have up to 4 channels to reach the destination by deploying 3 relay nodes in the network.

TimelyWare works as follows: When receiving a message and the respective deadline from the facility node, the TimelyWare node will select the most suitable channels, direct or using a relay, to send the message to the intended TimelyWare destination node. After receiving the message, the TimelyWare destination node will forward it to the facility or data center node that is connected to.

TimelyWare nodes follow the concept of plug and play where it is not needed any type of modification in the network in order to deploy them. They can be deployed in the network at any time. This is possible because the nodes periodically exchange information between them and are able to refresh their view of the network if new nodes arrive or exit.

In each node, TimelyWare is composed by several modules, as illustrated in Figure 4.2:

- The network parameters estimation module (NPE module) is responsible for measuring the one-way time (OWT), the round-trip time (RTT), the available bandwidth, and the network elements of each channel.

- The network adaption module (NA module) is responsible for adapting the estimated channel timeout and the estimated available bandwidth based on the latest measurements of the state of
the network.

- The **network prediction module** (NPred module) is responsible for predicting timing failures by analyzing and extracting frequent patterns from the timing failure table.

- The **channel matrix** is responsible for storing the data from all the channels.

- The **channel selection module** (CSel module) selects the primary and backup channels. The number of backup channels is dynamic and its calculated based on the deadline and the channel's delay.

- The **channel correlation module** (CCor module) calculates the correlation between channels based on the network elements present on each channel. The channels data is stored in the channel matrix.

- The **communication module** (Com module) is responsible for the communication between the TimelyWare nodes on top of the network.

The interaction between the modules works as follow. Periodically the NPE module will extract the intended network parameters samples and send them to the NA module in order to proceed to a proper estimation for each of the parameters. After the estimation process the estimated values will be stored in the channel matrix for the respective channel. Also the timeout value is sent to the NPred module. The NPred module will receive the current timeout value and decide whether a timing failure occurred or not. If so it will store the measurement timestamp in the timing failure repository. Periodically this module will proceed to the extraction of periodic events based on the information that is in the repository and update the channel's information. The CSel module will select the most suitable channels to send a message based on the information that is on the channel matrix, the correlation value between the channels obtained through the CCor module and the prediction provided by the NPred module.
The Com module was built using JGroups toolkit [3] to manage the nodes membership and to keep and update the respective views. Therefore TimelyWare nodes can be added to the network incrementally. This module also maintains an association between the logical address and the physical address of each TimelyWare node in the view.

4.2 Network Adaptation

This section will address how TimelyWare adapts to the current environment of the underlying network in order to maintain its goal of delivering the messages within the deadline. This section illustrates how the measurement of the network parameters works and how TimelyWare identifies the current state of the network.

TimelyWare periodically monitors the network in order to adapt to its current state. Based on the network parameters measured, it decides whether to take a pessimistic approach or an optimistic one. Both approaches refer to the number of channels that will be selected by the channel selection module. These approaches are discussed with more detail in Section 4.4. The network parameters measured by TimelyWare are presented in Section 4.2.1.

4.2.1 Network Monitoring

As previously mentioned, TimelyWare measures several network parameters. The network parameters measured are the following: RTT, OWD, available bandwidth and AS names. All of these parameters are measured frequently but each one has a different measurement periodicity.

To obtain RTT and OWT samples, TimelyWare nodes exchange UDP datagrams periodically with a small periodicity interval as the measurement does not impact the network performance. UDP was used instead of ICMP because it was discovered that ICMP was inaccurate in some cases due to load balancing mechanisms present in some routers, as discussed in [66].

In order to obtain accurate delay measurements, TimelyWare uses the Jnetpcap library [24] to obtain kernel timestamps therefore filtering the application delay. In this case Jnetpcap was used to decode captured packets in real time, thus allowing to obtain an accurate timestamp for when a message was sent and received.

TimelyWare obtains both OWD and RTT samples in one measurement, as illustrated in Figure 4.3, as this process also occurs when the TimelyWare nodes are sending application messages. When exchanging application messages the sending node will only know if the message reached the destination on time when it receives the respective acknowledgement. Therefore the ack comes with the time that the message was received at the destination.

The measurement works as follow: First, after sending the request message, Node A obtains the sent time and stores it. When the request reaches the destination node, Node B obtains the respective received time from the kernel. Then it sends the acknowledgement (ack) along with the request message received time. After receiving the ack of the respective message Node A will obtain the respective
received time from the kernel. For the RTT it subtracts the sent time to the ack received time. However for the OWD it’s a different procedure.

For measuring the OWT, TimelyWare needs the participant nodes have their clocks synchronized. Without synchronizing the clocks the OWD measurement would be highly inaccurate as each clock has its own drift rate. For example, the synchronization procedure can be done using NTP [59]. In the evaluation section we discuss how TimelyWare used NTP to synchronize the nodes clocks for the experiments. In order to synchronize properly each node extracts the clock offset, difference of their clock against NTP server’s clock, and the clock estimation error.

In order to calculate the OWD, Node A obtains its clock offset and estimation error when sending the request message. When Node B receives the request it also obtains these parameters. Then it also sends them along with the ack. When node A receives the ack, it calculates the OWD by adding each clock offset to the respective timestamp and then it subtracts the sending time to the received time. However if the total clock estimation error is higher than a certain threshold, the OWD sample is discarded in order to avoid errors due to outliers. The total estimation error is the sum of both estimation errors obtained by each node. The value for the threshold is discussed in the evaluation section.

The adaption of RTT and OWT can be done using TCP’s RTT estimation formula [65] or the Adaptare Framework [35]. TCP algorithm uses an exponential weighted moving average to estimate the round-trip time. The Adaptare framework uses the concept of coverage, the probability that a timing property will maintained over a time period, to calculate time bounds based on statistical properties of the current state of the environment. As both mechanisms are able to achieve good results, TimelyWare can be configured to use any of the two. TimelyWare uses a monitoring service to feed them the RTT and to get the respective estimated timeout. To be able to work with any of the mechanisms, TimelyWare interacts with a proxy service that implements the same functions as both of them. This abstraction level has the main advantage that it enables the selection of the mechanism to use at runtime and facilitates the adding of new timeout estimations mechanisms. After choosing the intended timeout mechanism the proxy service will be connected to it and it is ready to be used.

TimelyWare also monitors the bandwidth available in each channel. Although the accuracy of these measurements improves with measurement frequency, this frequency cannot be high to avoid congesting the network. This measurement is done through the pathload tool [51] due to its high measurement
accuracy, as discussed in Section 3.2.3. In order to use pathload, the receiving node can start only if the sender has started the tool. Therefore the TimelyWare nodes synchronize as follows: the sender node sends a request message to the destiny notifying it that it has started the pathload tool. After receiving the request, the destiny node starts pathload in receiving mode and the measurement starts. When the measurement is over the receiving node obtains the estimated available bandwidth value in the path and sends it to the sender node along with the ack message.

TimelyWare has also to assess the correlation between channels, although with a much lower frequency and this tends to be stable. Nodes assess this correlation by observing the Autonomous System (AS) traversed by each channel, using a variation of traceroute. Given the ASs traversed by two channels, the correlation between them is calculated using the Sorensen-Dice coefficient. Given two sets of ASs A and B, the correlation corr is given by 4.1.

$$\text{corr} = \frac{2|A \cap B|}{(|A| + |B|)}$$ (4.1)

To extract the ASs of a path, TimelyWare first obtains the IP of the routers that are present in the respective path. This is done using the paris-traceroute tool [21] as it can handle anomalies present in the network, caused by load balancing mechanisms in the routers, that traceroute [57] for example cannot. Then for each router IP it searches for the correspondent AS name. This is done using the whois tool [39] where for each IP address obtained the TimelyWare node performs a lookup to the multiple netwide directory services to obtain the respective AS of the domain owner. The main reason for choosing AS over IP is that the same router can present different IP addresses which would produce inaccurate results in the channels correlation calculation. With paris-traceroute it is possible to have multiple sets of ASs between two node due to load-balancing. In this case we calculate the correlation between all the the sets obtained during the measurement.

### 4.2.2 Network Stability Detection

TimelyWare monitors the channels to identify if they are stable or unstable. We use the term unstable to designate a network or channel with large variations in delay and bandwidth, where it is difficult to establish a trend. In this state the network becomes unpredictable. On the contrary, a stable network is characterized by having little variation in the parameters mentioned, being possible to establish a trend.

This stability detection approach is mostly useful for the available bandwidth estimation as its measurement frequency has to be low. Therefore we can have more information about the channel’s bandwidth between the measurement periods of the available bandwidth. This stability information could allow TimelyWare to adjust the timeout value for transferring larger messages.

TimelyWare identifies the channel state in three steps. First for each new extracted sample it calculates the RTT deviation, which is given by (4.2). The value of $\beta$ is discussed in the evaluation section.

$$\text{devRTT} = (1 - \beta).lastDevRTT + (\beta |lastRTT - newRTT|)$$ (4.2)
Then it averages the last $n$ devRTTs in order to calculate the stability ratio, given by (4.3). The $n$ value is discussed in the evaluation section.

$$\text{stability ratio} = \text{AVG}(\text{devRTTs})$$ (4.3)

Finally a channel is considered unstable when the value of this ratio is higher than a certain threshold. The threshold value is also discussed in the evaluation section.

### 4.3 Prediction of Instabilities Periods

This section is going to address how TimelyWare is able to predict when a channel is going to be unstable. Specifically, it is going to be explained the motivation behind this issue and the pseudo code of the prediction algorithm.

The main novelty of TimelyWare is the prediction of instability periods. The goal is to predict when a channel is going to become unstable, as this instability can affect message timeliness. The motivation for this mechanism is an effect we observed in several channels in different circumstances: high delay variations that occur with a certain periodicity, typically daily. An example is shown in Figure 4.4. It shows the RTT between a host in Singapore and another in Brazil during approximately 5 days (actually we observed the effect during 2 weeks, but the full graph was not easy to read). Everyday around noon there was a high increase in delay, which came down again about 12 hours later. We observed many cases like this one, often with much lower periods when the delay is higher. The issue is that when the delay passes from low to high, the RTT estimation takes some time to adapt, both when its done with TCP’s algorithm and with Adaptare. This increases with the period between RTT measurements.

![Figure 4.4: RTT samples collected over a week](image)

In this case we want to predict when these transitions periods, low to high delay, would occur and thus to avoid that path to deliver messages. Knowing that predicting the exact moment when these transitions periods will occur can be very difficult, we intend to find an interval where they may occur. Therefore we can avoid using the path during that specific interval of time. Moreover we want to find
intervals as short as possible, given the data available, instead of defining a fixed interval.

### 4.3.1 Prediction Algorithm

TimelyWare considers that there is a timing failure when the observed RTT is higher than the estimated RTT in a channel. Whenever it detects a timing failure, it stores the instant when it happened and the channel where it happened in a table. Periodically, e.g., everyday by midnight, it does prediction of timing failures.

The prediction algorithm is shown in Algorithm 1. Its objective is to search for time intervals when events (timing failures in our case) occur frequently. The key ideas of the algorithm are two. First, it searches for events that happen periodically in a large interval, e.g., a certain hour of the day; then if it finds them, it calls itself recursively to define thinner intervals in which there are still periodic events (e.g., a certain 30 min interval of the day, a certain 10 min interval of the day, etc.). The second idea is to measure the frequency of these events in all the intervals equally spaced, following the idea of frequency of the classical Apriori data mining algorithm [9].

#### Algorithm 1 Periodic Event Mining

```plaintext
1: procedure FINDPERIODICEVENTS(list, interval, cycle, start, stop, offset)
2:     intervals ← []
3:     subIntervals ← []
4:     cycleStartIntv ← offset
5:     while (offset = 0 and cycleStartIntv < cycle) or (offset ≠ 0 and cycleStartIntv < offset + intervalAbove(interval)) do
6:         newStart ← start + offset + cycleStartIntv
7:         if confidence(list, newStart, stop, cycle, interval) > confThreshold then ▷ If it is higher its because we found a periodic event
8:             belowInterval ← intervalBelow(interval)
9:             newOffset ← offset + cycleStartIntv
10:            subIntervals ← findPeriodicEvents(list, belowInterval, cycle, start, stop, newOffset)
11:            if subIntervals = null then
12:                intervals ← intervals U (cycleStartIntv, interval)
13:            else
14:                intervals ← intervals U subIntervals
15:         cycleStartIntv += interval / 2 ▷ Considering the sliding window
16:     return intervals
17: procedure CONFIDENCE(list, start, stop, cycle, interval)
18:     frequencyCount ← 0
19:     count ← 0
20:     for startIntv ← start; startIntv < stop; startIntv + = cycle do
21:         endIntv ← startIntv + interval
22:         if isEventInterval(startIntv, endIntv) = True then
23:             frequencyCount ← frequencyCount + 1
24:     return frequencyCount / count
```

When triggered in line 1, the algorithm receives a list of events, a time interval size, a cycle size, the dates of the first and last events, and an offset. The list in our case is the part of the timing failure table for a certain channel. The interval size refers to the size of the interval to look for an event, e.g., 30 minutes. The cycle size indicates the frequency of the events, e.g., daily or weekly. The offset tells from which interval the algorithm should start searching, e.g., the third 30-minute interval of the day (the function is recursive, when called form outside, this parameter is set to zero).

Starting from the given interval size and offset, the algorithm searches for events that occurred frequently in that interval (lines 5 to 14). This verification is made by the procedure confidence (lines 16-24) whose goal is to see if an event occurred in that interval in a high proportion of the cycles. This procedure returns a degree of confidence that there is a periodic event in that interval. The auxiliary procedure isEventInterval is responsible for verifying if an event occurred in the intended interval.
during a cycle. If the degree is higher than a threshold, the algorithm will repeat the search but with a smaller interval size given by the auxiliary procedure intervalBelow (line 8). This recursion (line 10) reduces the interval size of prediction and consequently improves the prediction accuracy as much as the data allows. This way, it is possible to obtain predictions with different granularities. Also, this algorithm uses a sliding window (line 14) so that it is possible to find events that occurred near an interval. For example the algorithm finds events that occurred between 1:30 a.m. and 2:30 a.m.

Consider the example in which the algorithm is called as follows: findPeriodicEvents(events, 60, 1440, 2014-10-05 00:04, 60, 1440, 2014-12-04 23:54, 0). The algorithm will search if there is an event that occurred every day in a 1 hour interval, from midnight to 1 a.m., from 1 a.m. to 2 a.m. and so on. If it finds that an event occurred periodically in an interval, the algorithm will search within that interval in sub-intervals, 15 minutes interval, on the next step.

In the end the algorithm will return a list of pairs indicating the start period and the duration of the periodical timing failures. For example it could return $\langle 3, 60 \rangle$ meaning that every day a failure can occur between 3 a.m. and 4 a.m. With this information, TimelyWare can avoid the channel on those periods.

The value for the confidence threshold is configurable, which means that user can specify when running TimelyWare.

4.4 Channel Selection

This section is going to address how TimelyWare selects the channels that offer the best probability to deliver the messages within the deadline.

To aid in the selection of the best channels to use, each node shares its information about channels with the others in order to have a complete view of the network. A channel is a logical connection between two nodes which provides an abstraction from the underlying network topology. When a node wants to send a message it can obtain a list of channels that reach the intended destination. TimelyWare does one-hop overlay routing, so channels can be direct (sender sends messages to the destination node) or one-hop (sender sends messages to another node that sends them to the destination node).

TimelyWare selects the channels based on their estimated timeout value and stability. Basically the approach for this case consists in sending the message through a base channel or primary channel while also sending copies through $k$ backup channels. The value of $k$ is calculated dynamically. The channel selection algorithm for small messages is shown in Algorithm 2.

To send a message, the procedure send receives the message, the intended deadline, the destination, the iteration and the number of messages sent (line 1). The iteration refers to the current number of attempts to send the message and the number of messages sent refers to number of messages that have been sent in previous iterations. The first time the procedure is called the iteration and messages sent is 0. This approach is iterative, meaning that in case of retransmission the channels used in previous iterations will not be selected as primary channel in the next iterations. For example if the first channel fails to deliver, then in the next iteration the next channel in line will be selected. In iteration 0 the algorithm selects the first channel as the primary channel. In the next iteration, 1, the algorithm
Algorithm 2 Channel selection for small messages

1: procedure SEND(msg, dest, deadline, it, msgSent)
2:   channels ← []
3:   backupChannels ← []
4:   primaryChannel ← null
5:   nBackupChannels ← 0
6:   channels ← filterUnstableChannels(OC[dest])
7:   if channels = [] or channels[0].OWD > deadline then 
8:     channels ← OC[dest, deadline]
9:   if channels = null and it = 0 then
10:      NOTIFY - FAILURE
11:   else if channels ≠ null then
12:      nLastResortChannels ← √channels.length
13:      for index ← 0; index < nLastResortChannels; index + = 1 do
14:         sendMessage(msg, dest, channels[i])
15:   else
16:      primaryChannel ← channels[it]
17:      sendMessage(msg, dest, primaryChannel)
18:      remainingDeadline ← deadline - primaryChannel.timeout
19:      nChannelsNextIteration ← OC[dest, remainingDeadline]
20:      attempts ← calculateAttemptsRemaining(channels, remainingDeadline)
21:      nBackupChannels ← getNLessCorrelatedChannels(primaryChannel, channels, nBackupChannels)
22:      for index ← 0; index < nBackupChannels.length; index + = 1 do
23:         sendMessage(msg, dest, backupChannels[i])
selects the second best channel as the primary channel and so on.

The algorithm works as follows: first the node obtains a list of channels that can reach the destination ordered, ascendent, by their respective timeout value (line 6). The goal is to select the channel that has the lowest timeout as the primary channel in order to ensure that in case of retransmission the algorithm still has enough time to send the message. Before selecting a channel, it filters out any channel that may soon become unstable (line 6). This is possible through the prediction algorithm, discussed in 4.3.1, which gives us the interval that a channel might become unstable. If there are no more available channels to use as base (line 7), the algorithm enters in the last resort mode (line 11 to 14). In this mode the algorithm selects the fastest channels (nLastResortChannels) to send the message. The value of nLastResortChannels is given by the square root of the number of channels (line 12) that can reach the destination within the deadline. Normally the more channels we use the higher is the probability that one will deliver the message within the deadline. However after a certain amount of channels the probability will not increase as the later channels in the ordered list do not offer good probability due to high timeout values or because they are unstable. In other words there is a limit of number of channels that after it the probability of delivery will not increase in the same ratio as the number of channels used. Therefore if the number of available channels is too high, for example 10, we want to use just a small number of channels that gives us the best probability to deliver in time while not stressing the network. And the square root function is the best approach that mimics the intended pattern. For example when the total number of channels is small we obtain almost the same amount of channels to use. However if we increase the total number of channels we obtain a sufficient number instead of using all or the majority which could lead to flooding. For example if we had 3 available channels than we would use 2 channels. If we had 10 channels then we would use only 3 channels. In this case we are using fewer channels but we are using the ones that give us the most probability to deliver the message in time. One remark is that if in the first iteration there is no channel fast enough to deliver the message (line 8) than the algorithm notifies the TimelyWare node that is not possible to send the message (line 9). However if this situation happens in later iterations the algorithm waits for the acknowledgement of one of the previous
attempts.

Nevertheless if there are available channels to be the base we select the fastest channel, based on the OWD, that has not been used as base (line 16). After selecting the primary channel, the node calculates the number of backup channels (line 21) based on the number of channels that can be used as backup (\(\text{channels.length}\)) in the current iteration, the number of channels that can meet the deadline in the next iteration (\(\text{nChannelsNextIteration}\)), the remaining attempts left until the deadline and the number of messages that have been sent until now. The remaining number of attempts (line 20) is calculated by forecasting how many primary channels can be used until the deadline. This is done by subtracting the channels timeout value to the remaining deadline until the difference becomes negative. The number of backup channels is given by Equation 4.4.

\[
\text{backups} = \frac{\text{channels.length} - \text{nChannelsNextIteration}}{\text{attempts}} - \text{msgSent} - 1
\]  

(4.4)

The goal is to use several channels if needed, but also to use them efficiently, e.g., to avoid using many in just one attempt if there are more attempts left. Therefore TimelyWare keeps monitoring the difference of available channels in this iteration and the next one. If the difference is high or the number of remaining attempts is low than we want to be aggressive. Therefore the number of backups should be high because in the next iterations we won’t have the chance to use some of the channels. However if the difference is decreasing slowly then the number of backups should be minimal, as we can use most of the available channels in the next iterations. One remark is that if the number of remaining attempts is zero then we just calculate the difference between the available channels in this iteration and in the next. The role of the number of messages sent is to moderate the value obtained by the difference. Also we subtract 1 to the value in order to take account for the selected primary channel.

The channels that are predicted to have timing failures until the deadline are not used. With the number of backup channels selected, TimelyWare then proceeds to select the channels least correlated with the primary (line 22) using the sorensen-dice coefficient.

When the primary channel’s timeout is triggered we proceed to the next iteration and repeat the procedure until there is no more time, reached the deadline, or no more available channels (line 9) or we receive an ack.

In conclusion, the goal is to use several channels if needed, but also to use them efficiently, avoiding using the majority of channels in just one attempt if there are more attempts left until the deadline.
Chapter 5

Experimental Evaluation

In this chapter we are going to analyze the results obtained while evaluating TimelyWare during several weeks in PlanetLab [7]. This chapter is structured in two sections. In the first section we are going to discuss values for the configurable parameters that TimelyWare uses. These parameters were described in Chapter 4. These parameters were used to achieve the proposed goals that were defined early in this work. In the next section we are going to compare the TimelyWare's results against other strategies that have the same goal in multiple types of environments. The comparison is mainly about the success delivery rate and the strategy’s efficiency using the network.

5.1 Configurable parameters

Multiple configurable parameters were described in Chapter 4 which explained TimleyWare’s architecture and how it was able to achieve the intended goals for this work. Those parameters are supposed to be configurable by the application that is going to use TimelyWare. In this section we discuss values for some of those parameters and the respective results when applying those values.

5.1.1 Network parameters measurement frequency

TimelyWare measures multiple network parameters in order to be able to understand the current state of the underlying network. TimelyWare monitors the one-way and round-trip delay, the available bandwidth and the ASs used to reach the destination. Table 5.1 shows the measurement frequency used by TimelyWare for each parameter. The column Network Parameter references the network parameter being measured by TimelyWare and Frequency column references the respective measurement frequency in minutes.

For the delays TimelyWare uses a minute frequency, i.e., it measures them every minute because these measurements do not exhaust the channels. For the available bandwidth TimelyWare uses a higher frequency because this type of measurement increases significantly the traffic in the channel. Therefore TimelyWare uses an hourly frequency which does not impact the network and still can be enough to extract information about the state of the network. For the ASs present in a path TimelyWare
Table 5.1: Measurement frequency for network parameters.

<table>
<thead>
<tr>
<th>Network Parameter</th>
<th>Frequency (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWD</td>
<td>1</td>
</tr>
<tr>
<td>RTD</td>
<td>1</td>
</tr>
<tr>
<td>Available Bandwidth</td>
<td>60</td>
</tr>
<tr>
<td>AS</td>
<td>15</td>
</tr>
</tbody>
</table>

uses a quarterly hour frequency mainly because routes do not change frequently therefore using a smaller frequency is not significant.

5.1.2 Clock Synchronization

As discussed in Section 4.2.1 the TimelyWare nodes synchronize their clocks in order to calculate the OWD. For this case the clock synchronization was done using NTP [59], which has the benefit of being available in many systems and does not need additional hardware and was already deployed in PlanetLab. In order to calculate the OWD, the nodes obtain their clock offset and estimation error when sending the request message or when receiving the request by communicating with the NTP daemon. The estimation error is also relevant as NTP uses RTT to deal with the synchronization uncertainty and as discussed in 3.2.2, the internet presents delay asymmetries. For the experiment we defined for the error estimation threshold a value of \(10\) ms. There have been multiple works that discussed the estimation error by NTP in WANs [60, 23, 8]. Generally the estimated error is around few dozens of milliseconds in WANs.

5.1.3 Network Adaptation and Prediction of Instabilities

To estimate the stability of the network TimelyWare monitors periodically the network. TimelyWare uses the latest 5 measurements made to extract information about the state of the network. This value was concluded through the analysis of several RTT measurements made during a month between PlanetLab nodes. It was possible to observe that with these latest measurements it was possible to conclude if a network was unstable or not. Also through these analyses we were able to define a value that classifies the stability of a network. When the network was presenting large variations in the delay we observed that for a stability ratio was of 0.5 using the last 5 measurements it was possible to classify the network as unstable.

For the prediction mechanism we configure the confidence threshold to a value of 75%. This value was selected after analyzing the timing failures obtained during the measurements between PlanetLab nodes and was possible to observe that this value was able to capture several intervals of instability resulting in multiple predictions.

5.2 Results

TimelyWare was validated on PlanetLab [7]. PlanetLab was selected for the experiment because it can provide a WAN environment while also providing control of each of its nodes.
The goal of this evaluation is to analyze the success rate and the cost of the TimelyWare strategy, for small messages, with varying deadlines. Also we intend to compare TimelyWare with other overlay strategies: Flooding, Primary-Backup (strategy used in the Italian power grid backbone), SORS [42] and Multi-Path which is a combination of [75] and [18]. The evaluation is composed of two major test cases and both were executed during 10 days. The goal of the first test case is to observe and compare the strategies behavior in a stable environment. The goal of the other test case is to observe their behavior when facing an unstable environment where it is possible to experience multiple timing failures due to network congestion for example.

The flooding strategy uses all channels simultaneously to send a message. The Primary-Backup strategy uses the direct channel supplied by the underlying network and in case of failure uses a backup channel, also supplied by the underlying network, if available. The SORS strategy sends the message through a direct channel and in case of failure it selects 4 random overlay channels as backups. The Multi-Path strategy uses simultaneously a direct channel and a random overlay channel to send the message. For this experiment we integrated the TCP timeout mechanism described in Section 4.2.1 to all of the other strategies in order to observe their behavior in case of retransmission.

For both test cases the message exchange frequency was 1 in each 5 min. In other words each node would attempt to send a message with a 5 minute interval. For example one node would attempt to send the message and then 5 minutes later it would attempt to the message but this time to another destination. For each attempt first the node would notify all the other nodes participating in the experience that it is its time to send the messages and then after sending the messages to all destinations it would notify them that it was over and the next node could start. This process is sequential, meaning that another node will start only after the previous has attempted to send the message to all the intended destinations.

5.2.1 Stable Environment

The goal of the first test case was to evaluate the strategies behavior in a stable WAN environment. The strategies were deployed in 7 PlanetLab nodes in the following regions: Ireland, Germany, Spain, Portugal, Rice (U.S.), Oregon (U.S) and Ontario (Canada). The values for the deadlines were 100ms, 200ms, 400ms and 600ms. These deadlines values were selected because we wanted to see how the strategies behave with strict values and also with values that give them a little breeding room. For each deadline the nodes sent messages to all of the other nodes. The receiving nodes replied with the time that the message took to reach to them. This test was executed during 10 days and each node exchanged only control messages. Table 5.2 shows the result of the test where the success rate refers to attempts in which the message was delivered within deadline over all the attempts and the channel usage ratio refers to the average number of sent messages per attempt.

As we can see, the Flooding strategy obtained the best success rate when the deadline was 100ms while all of the others, including TimelyWare, obtained similar results with Multi-Path being slightly above the rest. As the deadline increases all the strategies caught up to Flooding and obtained similar results.
Table 5.2: Success rate and channel usage ratio for each strategy in a stable environment.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Deadline (ms)</th>
<th>PB</th>
<th>TW</th>
<th>Flood</th>
<th>SORS</th>
<th>MP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100.0</td>
<td>83.81%</td>
<td>83.95%</td>
<td>87.01%</td>
<td>83.61%</td>
<td>85.28%</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td>99.86%</td>
<td>99.93%</td>
<td>99.49%</td>
<td>99.74%</td>
<td>99.74%</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td>100%</td>
<td>100%</td>
<td>99.91%</td>
<td>99.74%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td>100%</td>
<td>100%</td>
<td>99.82%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Efficiency</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 5.1: Number of msgs sent per attempt by each strategy.

Flooding was able to obtain the best success rate for 100ms because for this deadline it is almost impossible to have more than one attempt to send the message. This means that we do not have time to retransmit the message if a timing failure occurs. Therefore sending through multiple channels in one attempt would be the best option. However the number of channels that can deliver within this deadline is low thus TimelyWare can not use more channels as we would want to.

Figure 5.1 compares the channel usage by the strategies for each deadline. In this figure we can see better the difference between all the strategies in terms of message sent per attempt.

We can observe that Flooding is the least efficient as it uses all of the available channels to send a message and Primary-Backup is the most efficient using only one channel per attempt. We can also observe that TimelyWare is the only that varies the number of messages sent as it tries to adapt to the current condition. As the deadline increases TimelyWare approach becomes more optimistic and for 600ms we can see that it has the same efficient as Primary-backup strategy. Also it is possible to observe that TimelyWare had its least efficiency when the deadline was 200ms. This behavior occurs because as the deadline increases the number of available channels to reach a destination also increases. This gives TimelyWare more possibilities to choose. However as the deadline is still short, the number of attempts to send a message remains short. Therefore TimelyWare employs a more aggressive approach to improve the delivery probability.
Table 5.3: Success rate and channel usage ratio for each strategy in an unstable environment.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Deadline (ms)</th>
<th>Success</th>
<th>Efficiency</th>
<th>Success</th>
<th>Efficiency</th>
<th>Success</th>
<th>Efficiency</th>
<th>Success</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td>0%</td>
<td>1</td>
<td></td>
<td>0%</td>
<td>1</td>
<td>0%</td>
<td>1</td>
<td>0%</td>
<td>1</td>
</tr>
<tr>
<td>TW</td>
<td>80.6%</td>
<td>2</td>
<td>100%</td>
<td>5</td>
<td>99.69%</td>
<td>5</td>
<td>99.84%</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Flood</td>
<td>85.37%</td>
<td>6</td>
<td>100%</td>
<td>6</td>
<td>99.54%</td>
<td>6</td>
<td>99.22%</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SORS</td>
<td>82.44%</td>
<td>5</td>
<td>99.85%</td>
<td>5</td>
<td>99.23%</td>
<td>5</td>
<td>99.68%</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>34.24%</td>
<td>2</td>
<td>91.09%</td>
<td>2</td>
<td>99.54%</td>
<td>2</td>
<td>99.34%</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

5.2.2 Unstable Environment

For the second test case we wanted to evaluate the strategies’ behavior when facing an unstable environment where the probability of timing failure is high. Therefore we discarded every first message in order to simulate the intended type of environment. Table 5.3 summarizes the result illustrating the success rate and the channel usage ratio also. The same PlanetLab nodes and deadlines used in the first test case were used for this one. This experiment was executed during 5 days. The goal for this test is to see how the strategies can adapt to unstable environments. In other words is to see how effective their approaches are when facing message retransmissions.

In this case we observe that for the smallest deadline TimelyWare, Flooding and SORS obtained good success rate while Multi-Path (MP) and Primary-Backup (PB) did not. One remark for this test case is that for Primary-Backup the result obtained was expected. This is due to the fact that Primary-Backup only uses direct channels. However PlanetLab does not have multihoming thus there was only one direct channel for each destination and as we were discarding the first message then the result was always going to be the same, no delivery. Therefore Primary-Backup strategy is discarded for this comparison. As the deadline increases we can observe that the strategies converge to high success rates as they have time to retransmit the message when the timeout is triggered. In fact when for the largest deadline we can conclude that even with messages being dropped the strategies still manage to deliver the messages within the deadline.

As the strategies were able to obtain good success rates we focus now on the channel usage, which can give us more information about each strategy approach in terms of retransmission. Figure 5.2 shows in more detail the channel usage, number of messages sent per attempt, by each strategy for each deadline.

In terms of channel usage ratio TimelyWare was the most efficient between those which had good success rate while Flooding was the least efficient. For TimelyWare it is also possible to see the behavior described in the stable environment about the increase of messages sent per attempt when the deadline also increased from 100ms to 200ms. However this time we can observe that TimelyWare channel usage did not decrease right away as deadline increased more. This is due to the instability observed in the network as expected. As the deadline increased TimelyWare still was able to be one of the most efficient strategies. Another remark is that we can also observe the TimelyWare adaption mechanism in display as the number of messages sent per attempt was varying through the deadlines instead of just being a fixed value like SOSR or Multi-Path.
Just regarding TimelyWare, we can conclude that for control messages the expected result was obtained. TimelyWare was able to obtain high success rate while also being very efficient. For example for 100ms we were able to have the similar success rate as Flooding and SORS while being more efficient.
Chapter 6

Conclusions

In this work we proposed TimelyWare, a middleware that timely delivers messages over WANs in a reliable way with higher probability than alternative solutions for control traffic without the need to change the underlying network. TimelyWare is able to achieve its goal by taking advantages of overlay network and multihoming to establish multiple paths to reach a destination. Besides sending the message through a primary path, it also selects $b$ backup paths to send simultaneously. The value of $b$ is dynamic and is calculated based on the current environment of the network. The novelty of this work is in the ability to adapt to the current environment and predict the future state of the network as WANs presents different states in each period. The goal is to predict when the network will change its state thus it is possible to use the channels efficiently. This is done by extracting the channel’s temporal constrains.

We compared TimelyWare with other overlay strategies in the same conditions in PlanetLab. The results showed that TimelyWare is able to achieve similar success rate, in terms of delivering within deadline, as Flooding while sending less messages per attempt. In fact when the deadline increases we observed that TimelyWare has similar efficiency as Primary-Backup strategy which is the most efficient one. For unstable environments we observed that TimelyWare maintained its success rate while being the most efficient against those strategies which had similar success rates. Therefore we can conclude that TimelyWare was able to fulfill its requirements.

For future work we aim to evaluate extensively the benefits of TimelyWare event prediction algorithm and also improve the channel correlation algorithm adding statistical methods that are able to correlate more accurately the paths in terms of the probability to fail at the same moment. Another improvement would be in the algorithm to calculate the number of backup channels in order to improve the delivery probability and the channel usage efficiency. Also we aim to assure the timeliness requirements for larger messages like files for example by exploring the multiple available channels to reach a destination.
Bibliography


