
Università
degli Studi
di Ferrara

# UNIVERSITÀ DEGLI STUDI DI FERRARA

ENGINEERING DEPARTMENT

DOCTORAL COURSE IN "SCIENZE DELL'INGEGNERIA"

CYCLE XXXIV

COORDINATOR PROF. STEFANO TRILLO

---

# *Computer Networks In Tactical And Disaster Recovery Environments*

---

Scientific/Disciplinary Sector (SDS): ING-INF/05

**Supervisor:**
Prof. Cesare STEFANELLI

**Candidate:**
Dott. Roberto FRONTEDDU

**Co-Supervisor:**
Prof. Mauro TORTONESI

YEAR 2018 – 2021


Università degli Studi di Ferrara

# UNIVERSITÀ DEGLI STUDI DI FERRARA

ENGINEERING DEPARTMENT

DOCTORAL COURSE IN "SCIENZE DELL'INGEGNERIA"

CYCLE XXXIV

COORDINATOR PROF. STEFANO TRILLO

---

# *Computer Networks In Tactical And Disaster Recovery Environments*

---

Scientific/Disciplinary Sector (SDS): ING-INF/05

**Supervisor:**
Prof. Cesare STEFANELLI

(signature)

**Co-Supervisor:**
Prof. Mauro TORTONESI

**Candidate:**
Dott. Roberto FRONTEDDU

(signature)

YEAR 2018 – 2021

# Contents

# Chapter 1

# Introduction

Networks used to support Tactical And Disaster (TDR) operations are challenging communication environments in which insufficient capabilities of the network infrastructure limit information sharing. Nonetheless, these operations require a robust and predictable communication layer to support the concerted effort of several assets and operators that must collaborate to repel invading forces or rescue civilians after a natural disaster. Decision-makers want TDR operations to become more network-centric and follow the general trend of today's world to become more interconnected to take advantage and integrate globally available resources and innovative applications such as Commercial off-the-shelf (COTS) IoT devices, humans as sensors, GPS, social media, video/ audio over IP, and data transfer. TDR networks must bridge geographically dispersed and highly mobile groups of heterogeneous nodes operating in foreign and hostile environments. To do so, they must rely on precarious and opportunistically deployed network infrastructure since natural and fabricated disasters can significantly damage pre-deployed assets and force the use of a combination of heterogeneous radio equipment [1]. Network-centric communication paradigms must rely on transport protocols and group communication solutions to provide the required communication semantics while abstracting the transmission process from the complexities of low-level layers and dishomogeneous network interfaces.

Unfortunately, COTS solutions have historically presented degraded performance when deployed in TDR networks because characteristics of this communication environment violate many of the assumptions behind their design. This fact is unfortunate because the adoption of COTS solutions could reduce costs by leveraging economies of scale and can produce robust products by relying on widely adopted standards. Research in the fields of wireless networks and protocols has promoted the development and advancement of new solutions that promise better performance and resource utilization than traditional approaches under varying and challenging network conditions. Modern research efforts have also focused on improving the performance of TCP, studying solutions such as fair queueing, pacing, and segmentation offloading optimization and attempting to overcome several limitations that characterize this pervasive protocol. Some of these efforts have culminated in the design of new congestion control algorithms such as BBR [2], or brand new protocols such as QUIC [3].

Similarly, the field of group communication has also enjoyed significant interest. Recent technological advancements in data analysis and artificial intelligence have brought forward new group communication protocols specifically designed to support innovative applications such as data-pipeline monitoring, stream processing,

and event sourcing. While much of this progress targets enterprise-like networks, the internet, or other specific use cases, it is reasonable to think that some of these advantages will apply to TDR environments too.

Another complexity that characterizes TDR networks is that they present unpredictable and variable performance. TDR operations must span large geographical areas and rely on mobile nodes connected utilizing wireless radios. Morphological conditions and weather can impact the performance of wireless connections. To compensate for this, middleware and applications designed for these environments are generally highly configurable to support frequent node mobility, resource constraints, wildly heterogeneous devices, and strongly varying (and sometimes access denied) environments. Unfortunately, the extremely dynamic nature of TDR networks makes it impossible to optimally tune the behavior of communications middleware using predefined configurations or simple heuristic solutions and call instead for network-aware applications capable of continuously re-tuning to adapt to ever-changing operating conditions. However, accurately detecting the current network status and reacting accordingly still represents an open research question.

In this thesis, I present extensive experimental research on unicast and group communication protocols motivated by the fact that many improvements have been observed that may bridge the gap between enterprise and TDR environments. Significant parts of this thesis are dedicated to describing unicast protocols and the characteristics of group communication solutions from the point of view of using them in TDR environments. Since experimentation and analysis using realistic communication hardware capable of simulating TDR operations with high fidelity are important steps in the overall research and development process, this thesis also presents significant work conducted to create an open-source freely available, and militarily realistic emulated scenario that was used to conduct most of the experiments presented in this work.

The final contribution described in this thesis is Smart Estimation of Network StatE Information (SENSEI), a framework designed to provide network monitoring and adaptation in TDR networks. SENSEI can passively harvest network information such as exchanged throughput, bandwidth, and latency to infer the status of the network. SENSEI can then provide this information to other components, or it can use it to perform resource allocation or adapt other specifically designed communication middleware to improve the use of network resources and avoid congestion. In particular, this thesis presents SENSEI's architecture, a method to perform passive bandwidth estimation, and shows how SENSEI can be used in conjunction with other middleware to provide adaptive video streaming.

The rest of the thesis is organized as follows. Chapter 2 and 3 present respectively a functional description of TDR networks and operations the first, and the analysis of the requirements and challenges that characterize these communication environments the second. Furthermore, Chapter 3 also discusses relevant works.

Chapter 4 presents research conducted to evaluate the evolution of unicast protocols in terms of their applicability in constrained environments. Chapter 4 analyzes many TCP congestion control algorithms and alternatives and presents several experiments conducted under realistic and degraded conditions. Chapter 5 contains an analysis of several modern group communication solutions in the Anglova scenario, a military-realistic emulated environment that describes mobility patterns of a military operation conducted in the fictitious area of Fieldomont in Anglova.

In particular, this chapter analyzes several group communication solutions, presents to the reader characteriz-

ing features, and reports on several experiments conducted to evaluate COTS and custom solutions in terms of scalability, resiliency to packet loss, high latency, and low bandwidth. This chapter also discusses two implementations of Synchronized Cooperative Broadcast (SCB) used to power the Anglova scenario.

Chapter 6 presents SENSEI, a solution designed to provide network monitoring and adaptation in tactical environments. In particular, the chapter discusses SENSEI in terms of its architecture motivating design choices. The chapter also analyzes several related arguments, including the relation between SENSEI and communication middleware to implement network adaptation.

Finally, Chapter 7 presents future research direction and conclusions. The research effort presented in this thesis was conducted in collaboration with the Florida Institute for Human and Machine Cognition (IHMC), FL, USA, the NATO Science and Technology Organization (STO) IST-124 Research Task Group (RTG) on ”Heterogeneous Tactical Networks - Improving Connectivity and Network Efficiency”, and the United States Army Research Laboratory (ARL), Adelphi, MD, USA. The results of this work have been published or submitted in proceedings to several international conferences and journals.

# Chapter 2

# Tactical and Disaster Recovery Networks

This chapter introduces the concept of TDR networks and presents arguments that highlight the similarities between the tactical and disaster recovery communication environments. The first two sections describe tactical and disaster recovery networks from a functional perspective. The third section continues the analysis by presenting similarities between the two and discusses the use of COTS solutions in TDR operations, arguing that significant evaluation is required to conclude whether or not they can provide adequate performance in these complex environments.

## 2.1 Tactical Networks

Tactical networks are critical components of the Army's strategy to operate effectively and communicate in harsh and hostile environments [4]. These networks are highly heterogeneous, unstable, resource-constrained, and often targeted by disruptive activities conducted by adversarial forces. Their primary objective is to provide a robust and reliable communication layer capable of expeditiously delivering mission-critical information to soldiers and commanders in the field and decision-makers in the headquarters. The time-critical and resource-limited nature of tactical operations requires planning and optimal use of network resources to achieve the desired network performance while satisfying the need of participants to communicate. Tactical environments are generally resource-limited and access denied, these aspects create many challenges that network administrators and solution designers must overcome to achieve the desired functionalities. Nodes and devices operating in tactical networks can be limited in terms of processing and computation power, hardware may be battery-powered and consequently power constrained. Nodes' limited availability of bandwidth resources may significantly reduce the rate of communication. Signal jamming, denial of service attacks, and obstacles-rich environments such as forests and hilly locations may lead to intermittent network access. Moreover, the dynamic nature of military operations causes widely varied loads to be placed on the network by users and applications connected through wireless and ad hoc links in hostile radio frequencies. [5].

Figure 2.1 shows several nodes conducting a tactical operation in mountainous and vegetation-rich terrain. The scenario depicts two ground units, a warship, and an Unmanned Aerial Vehicle (UAV), providing sea and air support and surveillance. Vegetation and meteorological conditions can severely impact the satellite link that the warship and the ground units utilize to communicate. The ground operators can then expect intermittent

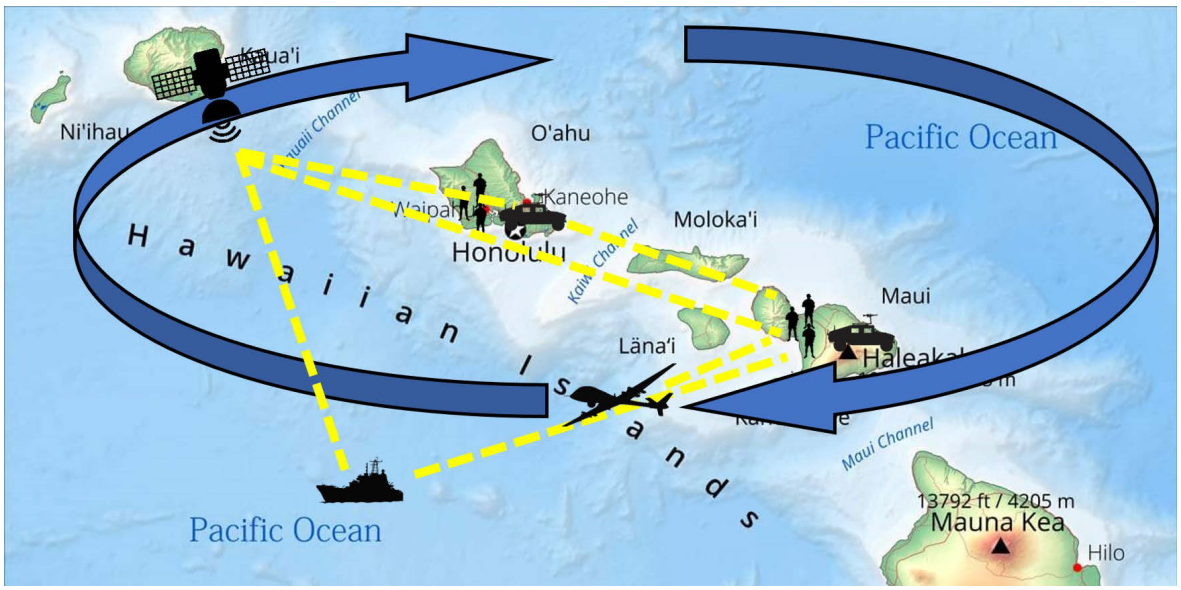



Figure 2.1: Overview of a tactical network deployment.

connectivity and various level of network performance ranging from reliable and high-throughput to severely impaired depending on whether or not physical obstacles block or weaken the wireless signals that support their portable radios.

## 2.2 Disaster Recovery Networks

Natural and manufactured disasters can be unavoidable and cause significant damage to human life and properties. Disasters such as earthquakes and tsunamis can be devastating and cause the structural collapse of network infrastructure and power grid that citizens and other entities rely on to conduct their activities. Rescue operators enact several endeavors to reduce loss and restore damaged areas. One of them is the deployment of battery or gas-powered disaster recovery networks. Rescuers can use these mobile ad hoc networks to broadcast information of the afflicted area to the outside world, to arrange, schedule, and organize rescue operations conducted by governmental and non-governmental organizations. These networks can help evacuees to return to their pre-disaster life by re-establishing network connectivity, giving a sense of normalcy [6]. Damage to the cellular network and power grid can be extensive, and consequently, it may take a long time before humanitarian efforts can restore affected communities. Common approaches to temporarily provide connectivity hinge on the use of satellite phones, portable base stations, and drones [7]. However, all these approaches need special hardware or other forms of preparation. Buildings and interference generated by surviving consumer electronics transmitters can further deteriorate this makeshift network environment [8].

Disaster recovery is a gradual process that generally starts with the involvement of state and non-governmental local authorities such as police, fire, and Emergency Medical Services organizations, the Red Cross, amateur radio operators, and other volunteers. However, when disasters overwhelm local authorities, specific federal agencies such as the Coast and National Guard can participate in the relief operations and assist civilian forces by bringing assets and expertise only available to armed forces. Moreover, it is not uncommon for armed forces to be deployed when disasters involve foreign countries under civil unrest or war, to collaborate, support, and protect civilian operators and assets. Military units are well prepared to work in challenging environments and

can provide protection, Command and Control, and Logistics adequate to support activities in damaged territories. To clarify this, we can consider the unrolling of a natural disaster within a war-torn urban environment afflicted by a cataclysmic earthquake that involves the participation of Doctors Without Borders supported by several Army units in the process of aiding injured civilians.

In this scenario, the city's network infrastructure has been devastated by an earthquake preventing communication. The pernicious effects caused by the natural disaster can severely impair the humanitarian effort [9]. Disasters can damage the physical components of the network infrastructure, such as antennas, routers, cables, optical fibers, or the power supply grid, fragmenting the urban networks in a multitude of fully-, and quasi-isolated sub-network with limited connectivity and unstable links. Due to these network conditions, ground operators must rely on mobile networks based on radios, satellites, or other wireless mediums to communicate.

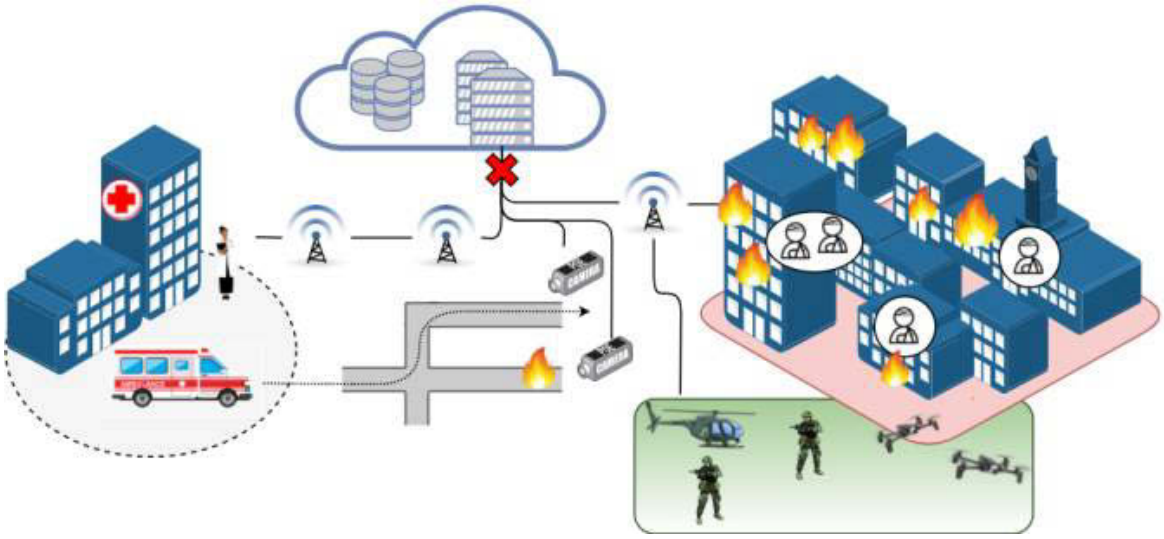


Figure 2.2: Overview of a disaster recovery network with multiple heterogeneous nodes.

Figure 2.2 shows a simplified but realistic deployment in which several civilian and military forces must collaborate to resolve a humanitarian crisis. The figure shows (on the left) a hospital connected through surviving cellular towers to several civilian and military operators (on the right). Near the disaster area, damage to network hardware has severed static infrastructure, forcing rescue operators to rely on networks provided through mobile hardware. In this example, the theatre of operation is a single city, but some operations span vast areas of hundreds of kilometers.

## 2.3 Tactical and Disaster Recovery Networks

Tactical and Disaster Recovery operations have several things in common. They are critical operations conducted by a plenitude of operatives that must cooperate to achieve a shared objective. They are highly dependent on the timely and reliable dissemination of information to multiple recipients to distribute orders, increase situational awareness, and prevent the loss of human life. They may not be motivated by the same reasons, but they share the hostile and unsafe environments in which they operate. Both operations are conducted in interference and obstacles-rich, resource-limited network environments. They both require a robust, reliable, and predictable network layer that must consistently be capable of sharing and receiving messages to create a

robustly networked force. A force of this kind can communicate and collaborate in the information and cognitive domains to create an improved information position for awareness and understanding. Unfortunately, the degraded nature of the communication environment that is characteristic of TDR operations thwarts this ideal.

In this thesis, I will use the term TDR networks to indicate the combination of heterogeneous communication resources inclusive of wireless, terrestrial, airborne, and space-based assets that together form the backbone of TDR operations. Several factors contribute to TDR heterogeneity. These operations are conducted in hostile or damaged territories and supported by headquarters and assets positioned in protected positions not afflicted by natural disasters or enemy activity. TDR networks must rely on several different technologies to support the integration of communication environments with distinct characteristics. For example, workstations located in static headquarters or naval assets can be wired together, but nodes at the edge must rely on wireless equipment to support connectivity on the move. In these situations, static infrastructure is unavailable either because it is absent, damaged/destroyed, or under enemy control. Furthermore, deploying network infrastructure in impervious terrains, such as forests, mountains, underground, or underwater, can require expensive and time-consuming heavy construction work, which is incompatible with the time and cost constraints that characterize ephemeral operations [10].

The challenges issued by next-generation TDR operations require a careful evaluation of unicast and group communication solutions. On one side, there is the desire to aggressively take advantage of COTS solutions to reduce costs and leverage economies of scale, but on the other side, TDR networks break many assumptions behind the design of commercial solutions that demonstrate degraded performance when deployed in network environments afflicted by limited bandwidth, long latencies, intermittent connectivity, abrupt variation in network conditions, or frequent node mobility. This fact is unfortunate because the adoption of COTS technologies enables reaping the benefits of economies of scale and facilitates and hastens the development and deployment of complex distributed applications by leveraging robust and widely adopted industrial standards and software components [11]. It becomes then imperative to understand the limitations associated with COTS protocols and devices to take advantage of them whenever favorable and otherwise develop alternatives or workarounds.

COTS applications are generally based on TCP and on hierarchical structures that have historically presented degraded performance when deployed in TDR environments such as excessive bandwidth utilization, decreased throughput, and other failures [12]. Nevertheless, researchers have been focused on COTS solutions to support edge environments, and quite a few improvements have emerged. This fact calls for the evaluation of modern communication protocols and middleware to verify the level of performance they can achieve when applied to TDR networks.

Moreover, considering the transitory and variable nature of TDR networks and that misuse can easily cause their collapse, there is a need for network-adaptive solutions capable of monitoring and adapting to network performance. Solution designed for this purpose must be resource frugal, reliable, and rugged since they must operate in resource-constrained and hostile environments. At a system-wide level, information about link capacity, bandwidth, stability, and delay, can be used to monitor links status to optimize overall end-to-end connection performance, update routing tables, or, in general, mitigate the effects of congestion. Communication middleware and protocols can use the information on path latency and bandwidth, to improve network utilization without being forced to rely on greedy congestion-control algorithms. Finally, at the application level, this

information can become an additional factor in the decision-making process that controls the request and publication of new information that can be opportunistically modified to adapt to the ever-changing circumstances. Network awareness can be a principal asset to support the situational awareness acquisition process by creating the basis to ensure that networked components can use the scarce network resources effectively.

# Chapter 3

# Requirements and Challenges

TDR networks present numerous challenges that solution designers and network administrators must overcome by communication and adaptation middleware to achieve reliable and predictable network performance. The technologies used to provide reliable inter-connectivity between geographically scattered nodes have specific characteristics that must be considered when designing and selecting solutions for these environments. For example, wired connections are fast and reliable but require physical connectivity. Satellite links provide high bandwidths and offer coverage of large areas but demand more time to deliver information since signals must first reach orbit before being relayed to their destination. Satellites may also present asymmetrical behavior in terms of the bandwidth they receive and send. Due to the necessity of base stations to connect with orbital satellites, cloudiness and other physical obstacles along the path can cripple these links reducing or preventing connectivity inside buildings, forests, underground, or underwater.

Tactical battery-powered radios used at the edge in obstacle-rich environments use VHF or UHF bands, are afflicted by frequent corruption-induced packet loss, low latencies, and limited range. These radios implement proprietary forwarding algorithms that can pass messages between intermediary nodes to compensate for disconnections caused by obstacles or out-of-range endpoints by implementing a dynamic overlay network between all the nodes belonging to a group. Network topologies of this kind, formed by heterogeneous radio equipment with a high mobility potential and connected through other links to other networks, are known in the literature as Mobile Ad-Hoc Network (MANET) [1] and the conditions that afflict these environments are classified as Delayed/Disconnected Intermittently-Connected Low-Bandwidth (DIL). MANET deployed in tactical environments reflect a scattered battlefield where soldiers and vehicles communicate between each other using links afflicted by unstable communication latency, varying channel capacity, high packet loss, frequent disconnection, and node mobility.

This chapter describes the challenges that software developed to operate in TDR networks must overcome to achieve reliable and predictable network performance. Compared to the previous chapter, this one is more technical and delves deeper into specific issues that characterize degraded environments. Sections 1 to 5 describe a range of typical issues spanning from limited bandwidth to security constraints. These issues are also summarized in Tables 3.1 and 3.2. Moreover, this chapter also contains related works to what the rest of this document will discuss.

| Challenge | Cause | Effects |
|---|---|---|
| **Limited Bandwidth** | Protocol Overhead | Congestion |
| | | Network Collapse |
| | Shared Network Resources | Congestion |
| | | Network Collapse |
| **Long Latencies** | Short Timers | Congestion |
| | | Connection Failure |
| | Multi-step Connections | Connection Failure |
| | | Slow Connection |
| | | Long Hold-ups |
| **Intermittent Connectivity** | Network Handover | Delay |
| | | Long Hold-ups |
| | High Churn Rates | Congestion |
| | | Network Collapse |
| | Loss-driven Congestion Control | Reduced Performances |

Table 3.1: Summary of challenges and repercussions of tactical networks characteristics.

| Challenge | Cause | Effects |
|---|---|---|
| **Abrupt Variations** | Bandwidth/Latency | Congestion |
| | | Network Collapse |
| | | Unused Resources |
| **Security** | Restrictions | Persistent Technological Erosion |
| | | Slow Development |
| | | Slow Updates |
| | | Unfaithful Testing Environment |
| | Encryptors | Prevent Communication |
| | | Reduced Performances |

Table 3.2: Summary of challenges and repercussions of tactical networks characteristics.

## 3.1 Limited Bandwidth

A primary challenge in TDR operations is that the ever-increasing quantity of data brought forth by the network-centric evolution of assets does not correspond to a proportional increase in terms of bandwidth that is available to transmit such information. This fact is due to the technological complexity of bridging edge and enterprise networks in TDR operations environments which cannot hope to match the growing data output of new generation devices. For example, operators cannot easily replace satellites without expensive and lengthy preparations. Moreover, the effective bandwidth provided by networking hardware used in MANET is often orders of magnitude smaller than the nominal values described in their specifications. This degradation can be attributed to various conditions such as high churn rate, interference from other radio devices, and poor connectivity. Furthermore, radio resources are shared between multiple devices exacerbating the effect of transport protocols overhead-induced starvation [13]. For example, the overhead that characterizes routing protocols commonly used in commercial networks may be an implicit limiting factor in the number of nodes that can be intercon-

nected [14].

Interconnected network-centric assets can also be sources of disruption in new and unexpected ways. For example, two operators connected respectively to a high- the first, and a low-resource subnetwork the second, could unbeknown to each other be consuming the same multicast video stream from the same source. As long as the traffic necessary to transmit a certain video quality is below a certain threshold, both subnetworks can sustain the video transmission without difficulty. A problem arises if the operator connected to the high-performance subnetwork requests the video at an increased quality. Since the source uses multicast to serve both operators simultaneously, it would have to increase the traffic circulating in both subnetworks to transmit the video in higher quality. The more constrained network may not have sufficient resources to handle this traffic, consequently saturating, and ultimately preventing communications.

Another challenge is that of adapting bandwidth requirements of COTS solutions to tactical environments. Common issues are the lack of sufficiently refined Application Programming Interface (API) or mechanisms to control the throughput of these applications in low-bandwidth networks. Some common causes of excessive data consumption are greedy data exchange mechanisms that lack sufficient logic to decrease the update frequency and non-trivial overheads of transport protocols and synchronization algorithms. This extra throughput often generated by inefficiencies and developers' ingenuity can quickly saturate bandwidth deprived networks or abuse scarce resources compromising global performance.

## 3.2 Long Latencies

COTS products and protocols are typically not designed to support communication between geographically dispersed assets. Commercial designers commonly assume that their products will communicate over the Internet or other equivalent high-performance networks, typically with short latencies under 100 milliseconds. Delay in multi-hop satellite connections used in TDR networks can easily reach dozen of seconds. Additionally, this problem is bound to be exacerbated by the advent of 5G networking which will eventually bring developers to expect latencies smaller than 1 millisecond. Applications that must operate in TDR environments should provide API to accommodate long delays. Undersized timeouts may prevent endpoints from establishing connections or waste resources by re-transmitting data supposed lost but just in transit. Long latencies can create problems for multi-step connection processes that require the exchange of multiple messages. In extreme cases, transient resources may systematically fail to complete these processes before moving out of radio range. The intermittent nature of tactical links exacerbates this problem by forcing endpoints to reconnect frequently.

Long latencies can complicate assessing network state. For example, TCP has a default interval of 300 seconds before considering stale a connection without exchanged traffic. Because of that, it is not uncommon for solutions that use this transport protocol to develop a status detection algorithm on top of it. A common approach is to exchange periodically small messages called heartbeats that measure the round trip time as the interval between the send and acknowledgment times of a heartbeat, and packet loss as the difference between sent and received heartbeats. These mechanisms are frequently troubled by the same lack of configurability that characterizes COTS solution. Consequently, they could generate excessive traffic if the exchange periods are too small or fail to recognize active connections if the maximum timeout is smaller than the network latency.

The two following examples highlight the practical implications of failing to provide adequate API. The TCP three-way handshake implementation of Windows 2008 R2 and Windows 7 had a maximum timeout of 21 seconds. When used to connect endpoints separated by more than 12 seconds of delay, missing even one of the three handshake messages would cause the connection process to fail. The second example stems from our experience in datagram security. Until 2021, the DTLS implementation of the Bouncy Castle security library did not provide a method to override the default maximum handshake timeout preventing communication middleware from establishing encrypted connections in delayed networks.

In general, solutions for TDR environments should account for long latencies, provide API to control maximum timeouts, and avoid implementing blocking synchronous behaviors that could reduce responsiveness by waiting for replies from remote resources.

## 3.3 Intermittent Connectivity

Intermittent connectivity is a typical occurrence in MANET since nodes connected through wireless technologies move in and out of range in obstacle-rich environments.

Intermittent connectivity can have several negative consequences. For once, loss-based congestion control transport protocols interpret packet loss as congestion and react by reducing transmission rates. This assumption is frail in tactical environments because loss can also be an intrinsic part of the nature of a wireless communication channel that can be unrelated to saturation and resource contention. Moreover, MANETs are often used to connect mobile assets that navigate through obstacle-rich environments, and transport protocol should not interpret packet loss caused by this fact as a signal to reduce transmission rates.

TDR solutions should expect to be disconnected for long periods and should minimize the time and number of packets required to reconnect. By doing that, solutions can prevent wasting precious network bandwidth and avoid the pitfall discussed in the previous section of needlessly lengthening the reconnection process in the presence of long delays. Ideally, TDR endpoints should preserve connection information on disconnections and implement simplified reconnection and sync processes based on this information.

## 3.4 Abrupt Variation in network conditions and topology

A most daunting characteristic of tactical environments is that they can be turbulent and chaotic [15], change unexpectedly because of enemy activity, weather, and node mobility, and cause extensive performance fluctuations in terms of latency, packet loss, and maximum throughput.

Solutions capable of sensing and adapting to network changes are necessary to opportunistically take advantage of available resources or reduce consumption to prevent starvation. Opportunely designed APIs can increase flexibility, support modifying the behavior of transport protocols, and the implementation of applications and middleware logic that can efficiently summarize and aggregate data. Communication middleware should use receivers' interest to prioritize messages and shape informative content to adapt it to the network status. For example, a video streaming application could dynamically change the quality of a live feed based on available bandwidth or operators' interest. Data fusion algorithms could change the coarseness and delivery periods of

sensor data, altering transmission formats to improve network utilization and modify redundancy and availability to increase the chance that critical messages are promptly delivered.

Since restrictive Quality of Service (QoS) policies typically require increasing network footprint to provide more guarantees, it is critical to pair levels of service and data relevance accordingly to avoid wasting resources. For example, positional reports and calls for fire have very different requirements. The firsts are small messages sent regularly. These messages contain geographical coordinates that describe units' positions over time. The periodic nature of these messages makes them quickly lose value over time and removes the need for reliability. Conversely, calls for fire are messages used to describe offensive maneuvers. To avoid casualties and increase effectiveness, they need to reach their intended destination reliably and quickly.

It is worth noting that the data priority of messages can change over time depending on the mission status and on what operators need from the network at a given time. For example, communication middleware should dynamically change the priority of video sources over other traffic when operators or analysis tools are consuming them and should otherwise reduce their transmit rate to save resources. Several types of information lose importance over time and should lose priority accordingly. Moreover, critical information can get queued in transmission buffers behind large sequences of unimportant data. Communication middleware should provide interfaces to retrieve these messages and expedite their delivery over other less critical messages.

## 3.5 Security

Security is a chief aspect of developing solutions for tactical environments. The sensitive nature of TDR operations calls for stringent security measures that complicate software development and increase the time required to deploy updates and fixes.

Lengthy authorization procedures force new software solutions to interact with obsolete technologies and be subject to inexorable technological erosion. Restricted environments forbid several common operations such as running applications with administrative privileges or the use of unaccredited libraries and operating systems. Inline network encryptors designed with the objective of preventing breaches of information between secure and insecure networking spaces strip transport protocols of custom headers preventing the adoption of new variants or causing unexpected network failures [16]. Prohibitive costs and security concerns create significant differences in the characteristics of test and deployment scenarios, complicating the process of verifying software functionalities.

## 3.6 Related Works

Several researchers have proposed improvements to TCP congestion control algorithms or offered new protocols, but few have investigated currently available solutions in constrained networks.

In [17], H.L. Gururaj et al. examined many TCP congestion control algorithms and compared the throughput and packet drop rates between six nodes using a shared simulated link. The experiment was run with a fixed bit rate but no information was released about latency and packet loss. In [18], T. Yanping et al. evaluate three custom improvements to TCP congestion control in an ad hoc network simulated using NS2. The authors

focused their observations on fairness and throughput. In [16], C. Richard et al. discussed how transport protocols interact with the different characteristics of military networks. The authors also compared a number of TCP congestion control algorithms at varying levels of packet loss. In [19], A. R. Urke et al. described a test environment made by a hybrid network consisting of an emulated satellite and radio leg used to connect two clusters of hosts that exchanged military-like traffic. The authors used this testbed to evaluate the competing flows of several TCP implementations. The emulation was conducted using Netem, Dummynet, and Vyatta. Netem was used to simulate delay and random loss, Dummynet to control bandwidth, and Vyatta, to create an IPsec encrypted tunnel. In [20], M Bell et al. argued that to improve the performance in MANETs and to distinguish between congestion and link error or loss, it is necessary to process explicit congestion notifications from the network. The authors compared this approach to TCP-Reno, a loss-based congestion control algorithm inherently fated to underperform in MANETs due to its loss-based detection of congestion.

Similar to the experimentation for unicast protocols, we were able to find only limited research that evaluated modern group communication mechanisms in realistically emulated network environments. In [21], A.V. Terkhedkar et al. delivered a comprehensive analysis on publish/subscribe solutions focusing on security and threat analysis. In particular, the authors studied several protocols focusing on their approach to security and threat mitigation.

In [22], A. Scaglione et al. described an initial approach to cooperative broadcasting at the symbol level, called Opportunistic Large Array (OLA). Cooperative broadcasting is also described as Barrage Relay Network (BRN) in [23]. BRN and synchronized cooperative broadcasting are similar as both use autonomous cooperation for time-synchronized nodes and a TDMA structure. In [24], J. Grönkvist et al. showed the performance of synchronized cooperative broadcasting dynamic scheduling. In [25], A. Komulainen et al. showed a comparison of the network broadcast capacity obtained by synchronized cooperative broadcasting and traditional TDMA-based schemes. Synchronized cooperative broadcasting was shown to be more efficient in mobile networks unless the network was large and dense. These articles provided important results and considerations at the base of the implementation of the realistic emulation scenario that will be discussed in Chapter 5.

Significant prior work on network monitoring and adaptation has focused on using statistics such as congestion level, packet-loss rate, and available path capacity to estimate metrics that affect applications' performance in constrained network environments. Related works did not, however, address the problem of determining the underlying technology powering communication, generally assuming to already possess that information, or ignored this problem altogether [26] [27]. Identifying the technology that powers communication can help understand why the network is not performing as expected and simplify troubleshooting. Extracting this knowledge is not trivial because security barriers often prevent applications from querying network devices.

Several strategies can be employed to reach network awareness. Operators can monitor links of interest by accessing/logging into devices [28], or via Simple Network Management Protocol (SNMP) [29], an Internet standard protocol that network managers can use to query and issue requests to compatible network devices such as switches and routers. Yet, SNMP has a non-negligible network footprint, and while researchers tried several approaches to reduce its overhead to increase portability to resource-constrained environments like tactical networks [30], proposed solutions typically require standard-breaking changes to the protocol and the deployment of custom agents and managers that severely limit their adoption. SNMP information is also not

widely available to applications. While network administrators may query devices under their control, they can't generally access monitoring devices outside their domain. Still, a majority of the solutions proposed to monitor degraded environments are based on SNMP and are in general not passive.

In [31], G. Kuthethoor et al. evaluated SNMP performance in tactical environments and observed that metadata associated with reports often required more space than the informative content itself. Moreover, SNMP does not support Multicast with evident overhead implications in the polling phases [31]. In addition, solutions designed for TDR environments should be bandwidth conscious and implement sophisticated algorithms capable of summarizing and aggregating data to reduce their network footprint. More detailed information should only be shared when needed or requested by network operators.

In [26], T. Chen et al. presented an SNMP-based solution that shares information on a per-flow basis and consequently cannot aggregate and consolidate multiple flows into a single metric to save traffic. Moreover, the solution only focused on reducing congestion and did not characterize links between endpoints to gain more insight into the communication status. In [27], S. Peng et al. used Network Management Station (NMS) (another SNMP based solution) to retrieve network statistics. NMS provided resource allocation by controlling other specifically designed applications and did not expose its assessment of the network state. Yet, solutions designed for TDR should share monitoring information with other smart middleware to increase network awareness and decentralize resource allocation.

In [32] and [33], the authors discussed link identifiability and algorithms to calculate the minimum amount of sensors needed to identify all the links in a network. The authors claimed that as a communication network grows, it becomes increasingly harder to place monitors for each link, a fair conclusion considering the configuration deluge associated with custom network environments. In [34], the authors presented NetNORAD, a system that treats the network like a black box and troubleshoots network problems independently of device polling. NetNORAD identifies relevant changes in network conditions using an algorithm based on percentile variations. NetNORAD relies heavily on active probing, making this solution impractical in resource-constrained scenarios characterized by rigid bandwidth constraints. In these cases, it may be more suitable to use passive approaches and instead piggyback monitoring on non-monitoring data packets. For example, it is possible to extract the Round Trip Time (RTT) by monitoring ICMP and TCP packets of ongoing connections. In [27], A.S. Peng et al. presented AutoDRM, another mechanism based on SNMP designed to manage shared resources without human intervention. AutoDRM centralization makes it impractical in TDR environments often access denied and compartmented.

Bandwidth estimation is an important part of network monitoring and several researchers have proposed algorithms designed to estimate this metric. Variable Packet Size Probing (VPSP) [35] estimates it across a set of path hops by sending groups of packets with different Time to Live (TTL) values. The sender forces these packets to expire and uses ICMP Time Exceeded messages to estimate the RTT at each hop. The minimum RTT at hop $i$, $RTT_i$ can be expressed as:

$$RTT_i = a + \sum_{k=1}^{i} (\frac{S}{C_k})$$

where $a$ is the delay up to hop $i$, and each member of the sum, is the contribution at hop $k$ with $S$ and $C_k$ being respectively the size of the packet and the capacity of the hop. From this formula, the slope of the linear

interpolation of the minimum RTT measurements is the inverse of the capacity estimate at that hop [28]. VPSP can suffer significant measurement error. One common reason is that store and forward switches and other network boxes frequently do not generate ICMP Time Exceeded messages for security reasons. Finally, some classes of network devices, including proxies and radios, might ignore or modify the value of TTL fields in IP headers, thus influencing the estimation accuracy of VPSP.

Another technique is based on Packet Pair dispersion (PPD) [36]. This approach estimates a path's total capacity by sending packet pairs of equal size towards a receiver. Each pair is made of two or more packets sent back to back (in the latter case the pair is called "packet train"). This method uses the delta between the time the last byte of each packet has been received to extract link capacity. With no cross-traffic interference, the receiver can extract that metric using the formula:

$$D = \frac{S}{C}$$

where D is the delta between packets, S is the size of the packets, and C is the capacity of the link. Packet dispersion techniques are especially impacted by the presence of cross-traffic that can either decrease (the first packet delayed more than the second) or increase (the second packet delayed more than the first) the measured delta, thus affecting the accuracy of bandwidth estimation. Several strategies have been developed to filter cross-traffic error such as sending trains of packets with different sizes and using statistical or machine learning-based models (including linear regression, Kalman filters [37], neural networks [38], and measurement repetition) to filter out bad samples.

Self-Loading Periodic Streams (SLoPS) [39] is a technique based on sending a certain number of packets of equal size until the receiving rate reaches the channel bandwidth. There are several similar approaches based on this methodology: ASSOLO, TOPP [40], Pathload, PathChirp, FEAT, and BART, see [35] for a comprehensive analysis. In [41], J. Strauss et al. present SPRUCE, an (active) available bandwidth estimation tool that sends a train of UDP packets between two monitoring endpoints while measuring the delta between probes. SPRUCE's users must specify a capacity that the solution then compares with the available bandwidth estimated from the inter sending time between probe packets through:

$$AB = C * \frac{\Delta t_{out} - \Delta t_{in}}{\Delta t_{in}}$$

where C is the capacity of the bottleneck, $\Delta t_{out} - \Delta t_{in}$ is the time to transmit the cross-traffic, and $C * \frac{\Delta t_{out} - \Delta t_{in}}{\Delta t_{in}}$ is the rate of the cross traffic. In [42], Oshiba et al. present PathQuick, an estimation algorithm that generates a packet train in which each packet is placed at an equal time interval and has its size linearly increased as the packet sequence proceeds. PathQuick was developed with the idea of being used right before transmission to optimize the subsequent delivery process and avoid the long time necessary by other algorithms to produce an estimation. Each packet includes the time it was sent so that the receiving node can use it to estimate the available bandwidth through:

$$D = \frac{S}{C}$$

This approach requires accurate network synchronizations between the communication endpoints.

Probing can be either active or passive. Active solutions generally provide better accuracy at the cost of introducing additional network traffic, a tradeoff that can be problematic in bandwidth-constrained or high-latency

networks. In light of this, passive bandwidth estimation techniques may be preferable. For example, QoS-AODV [43] infers the link capacity by taking the lowest ratio between received and sent packets for each one-hop neighbor; each node then shares its result with other nodes. Contention–aware Admission Control Protocol (CACP) [44] is a solution designed for 802.11 ad hoc networks that can estimate the bandwidth as the idle channel ratio multiplied by the channel nominal capacity. Other solutions such as [45], [46], and [47], extend this by considering collision, re-transmissions, back-offs, and node mobility respectively. All these algorithms are either active or rely on unreliable passive measurements. Ideally, a communication middleware could extract information from traffic sent for other purposes and perform bandwidth estimation without injecting any extra traffic. This could reap the benefit of PPD and variants without one of their most considerable drawbacks: significant overhead.

One highly desired feature to support TDR operations consists in providing video streaming to bandwidth depleted nodes. In [48], H. Pinson et al. presented subjective research on the characteristics that a video feed should have to be acceptable from the perspective of emergency first responders. In their article, the authors identified several metrics to evaluate video streams: The One-Way Video Delay, Frame Rate, Luma Image Size, Lossless Coding And Transmission, Codec Type And Bit Rate, and the Packet Loss And Error Concealment, these characteristics are summarized in Table 3.3.

| Characteristic | Description |
|---|---|
| One-Way Video Delay | Length of time taken to send a video through the entire video system |
| Frame Rate | Rate at which a video system can produce unique consecutive images |
| Luma Image Size | Black and white proportion of the video picture |
| Lossless Coding And Transmission | Data loss before and after compression |
| Codec Type And Bit Rate | Amount of information that the video codec outputs into the network, excluding network overhead |
| Packet Loss And Error Concealment | Fraction expressed in percentage of the packets lost by the network and a mechanism to counter this effect |

Table 3.3: Several metrics to evaluate video streams for TDR operations.

In [49], S. Pudlewski et al. discussed many video encoders and networking protocols for wireless video streaming to introduce the reader to the generalities of video streaming and analyze different strategies used to improve it. The authors also argued that streaming systems are limited by the following constraints: limited data rates (this caps the amount of bandwidth that can be freely used by video without congesting the network), limited power (users are quickly moving from desktop to mobile applications which are battery-powered), channel conditions (in terms of packet loss or other elements that may affect the stream), and network complexity (applications have generally low visibility of the network topology). In [50], J. Nightingale et al. presented a comprehensive analysis of H.265. They also introduced a service capable of harvesting metrics such as bandwidth, end-to-end delay, and packet loss ratio, calculated using a cooperative mechanism that had clients

periodically report received statistics to a server for each given video stream. This algorithm applies forward error correction to ensure the delivery of the most important components of the video. The authors also introduced a selective dropping scheme to limit the throughput in case of bandwidth constraints.

Not much literature focuses on multiple concurrent video streaming or node mobility. Proposed solutions do not take into account that multiple streams could be active at the same time and only considers the quality of single streams. They also do not take into consideration that in TDR operations there is a need to implement opportunistic behaviors and to cache and forward videos to overcome dynamic network fragmentation.

# Chapter 4

# Experimental Evaluation of Unicast Solutions in Constrained Environments

This chapter describes prominent protocols used in unicast communications, dedicating considerable space to TCP and to several congestion control algorithms developed for it. TCP is one of the most pervasive transport protocols and as such, it has been under extensive scrutiny over the years. This chapter also examines many alternatives developed to overcome specific limitations of TCP such as Stream Control Transmission Protocol (SCTP), which tries to overcome Head of Line Blocking (HoLB), Quick UDP Internet Connections (QUIC) a protocol designed to replace TCP in Google's networks, and Mobile Sockets (Mockets), a communication library that IHMC developed with the specific purpose of overcoming TCP's limitation in tactical environments.

## 4.1 Unicast Communication

Unicast communication consists of a one-to-one transmission between a transmitter and a receiver generally achieved through some application library capable of providing basic QoS classes of services (reliable/sequenced) and as a way of abstracting from heterogeneous hardware. TCP is arguably the most commonly used transport protocol making it a great candidate to study and compare to other alternatives. Moreover, several problems have historically plagued TCP, fueling improvements and the development of alternative solutions.

In the late 1970s, the Defence Data Network (a computer networking effort of the United States) developed TCP intending to enable communication using static and wired connections. Since then, TCP has been used in several network environments, consistently drifting towards mobile wireless networks. Due to the different characteristics of the original and current scenarios, researchers started observing several cases in which TCP was under-performing [51] [52] and consequently proposed and developed variations of the original TCP and competing solutions.

One major cause of performance degradation was that early congestion control implementations used packet loss to detect congestion. This assumption can be quite flawed in radio environments. While congestion is one of the causes of packet loss, insufficient storage space, bandwidth capacity, lack of processing power, and network errors can too cause this phenomenon and happen much more frequently in radio environments

[53]. This breach of assumption created several situations that made TCP underperform. Moreover, loss-based congestion control algorithms have also been proven to underperform in networks characterized by a high bandwidth per latency product. TCP is affected by numerous other problems such as HoLB, buffer-bloat, and Network Handover Driven Delay. Even considering all these issues, TCP is still the most used transport protocol in the world, and plenty of improvements have been proposed to address its limitations. Improvements to TCP have to be limited to the congestion control algorithm. Widely adopted network boxes developed to improve TCP performance have come to expect a specific packet format that, if not respected, cause these boxes to behave erratically or silently drop packets. This explains why research has mostly focused on congestion control algorithms or alternative protocols.

## 4.2 Classic TCP Problems

Several notorious problems have historically plagued TCP, the most infamous ones being Bufferbloat, HoLB, Network Link Handover, and Half Open Connections.

Bufferbloat [54] is a term coined to describe the negative interaction between TCP greedy congestion control algorithm that increases throughput until it detects loss and the large buffers that are part of modern networks. Large buffers are necessary to not under-utilize communication channels, yet, they can distort the ability of TCP to correctly estimate the available bandwidth resulting in packet loss at the slowest link. Additionally, these large buffers require time to empty, thus increasing the average latency of the network. This problem will become more prominent in the future with the advent of 5G. This technology is characterized by very low latencies that will increase the impact of the degradation introduced by large buffers. Historically, Advanced Queue Management (AQM) [55] has been the research field focused on improving routers to allow them to detect bufferbloat and implement countermeasures such as selectively dropping packets or reducing the throughput of selected flows. It is worth noting that commonly used bufferbloat countermeasures may not work very well in tactical environments due to the long and variable latencies associated with satellite links.

HoLB is a phenomenon that happens in sequenced transport protocols used to perform independent concurrent requests through a single connection. Sequenced protocols must deliver stream fragments reliably and in order. It follows that missing a fragment of one request will have the effect of stalling all the others until that fragment is delivered, adding significant delay to flows that should be independent of the missing fragment [56].

Network Layer Handover (NLH) is a phenomenon that happens when a node changes its IP address. NLH can happen for several reasons, such as the detection of an IP conflict or the node moving from an access point to another [57]. NLH is a source of delay because many protocols and software use IP addresses to identify endpoints. Changing IP then becomes a significant source of disruption that forces operators to re-configure applications and re-initialize connections with associated delays. NLH effects are worse in networks characterized by huge delays because multiple packets, with consequent delays, have to be exchanged at each reconnection phase even if the NLH happens in a brief interval of time.

There are many approaches to circumvent NLH which add different levels of complexity. The most simple and slowest one is to terminate the current connection and create a new one upon detecting an IP change. Better solutions may try to hold the old connection until a new one is established, but that is only possible

when multiple interfaces are pre-configured with expected addresses and the software subject to the handover is sufficiently refined.

Half-Open Connections are connections that have not yet been recognized stale. Since TCP only detects dropped connections at the sender side, receivers have no way to recognize those other than waiting for long timeouts. In extreme situations, these connections can represent a significant drain of resources. Moreover, attackers can exploit them to destabilize receivers.

## 4.3 Congestion Control Algorithms

Congestion control algorithms can be classified depending on whether or not they trigger congestion events on packet loss (loss-based congestion control algorithms) or delay (delay-based congestion avoidance algorithms).

Delay-based congestion avoidance algorithms use the variation of the average queuing time on a path to measure bandwidth. Arguably, delay-based algorithms have several advantages over their loss-based counterparts. In homogeneous networks, delay-based algorithms can provide better bandwidth utilization avoiding loss and making instantaneous transmissions more stable than standard TCP. Another advantage is that competing delay-based algorithms' flows that traverse the same route can fairly share bandwidth since they all observe the same average queuing delay variation.

Nonetheless, delay-based congestion avoidance algorithms hold many disadvantages that have hindered their adoption. First, when they must coexist with other protocols in heterogeneous networks characterized by multiple, asymmetrical, and different paths, these congestion algorithms tend to take advantage of far less bandwidth than delay-based counterparts [52].

A second issue is called Persistent Congestion Problem [52] which causes new flows to overestimate queuing delay. Once a group of flows is in equilibrium, new flows will assume that the current queuing delay characterizes the network. This is not correct because all the competing flows should rebalance their throughput considering new flows to achieve fairness. Delay-based algorithms also have problems in asymmetric links because delays on the receive path influence the measure that the sender will use to decide how much to transmit [52].

The following is a summary of notorious congestion control algorithms compiled to complement results and observations in future sections. The interested reader can find a detailed analysis of these protocols and additional ones in [53].

### 4.3.1 Tahoe, Reno, New Reno, SACK, Vegas, CTCP, and Westwood

Tahoe can be summarized by describing its three phases: Slow Start, Congestion Avoidance, and Fast Retransmission. Slow Start is executed at the beginning or when packet loss is detected. In this phase, the Congestion Window (cwind) is exponentially increased by two packets each time an acknowledge is received until the cwind reaches a certain configured Slow Start Threshold (ssthres). The cwind is a variable that limits the amount of data that TCP can send to a receiver and that also identifies the number of packets sent but not acknowledged.

Once ssthres is reached, Tahoe enters Congestion Avoidance, increasing the cwind at a much slower and steady pace of $\frac{1}{cwind}$ per acknowledge to improve fairness towards other flows and avoid system instability.

After a loss, Tahoe can take advantage of the fact that TCP acknowledgments are cumulative, speed-up recovery, and avoid having to wait for the pipe to drain, by assuming that each time it receives 4 duplicate acknowledgments for packet i, packet i has arrived, i + 1 got lost, and i + 2, i + 3, and i + 4 have likely arrived too.

Once that situation is detected, Tahoe enters Fast Retransmission where it sets the ssthres to $\frac{cwind}{2}$, and cwind to 1 Maximum Segment Size (MSS). The MSS is a parameter of the TCP connection that specifies the maximum size of a segment which in turn consists of a header and a data section. After that, Tahoe resets to Slow Start.

Reno [58] improved on Tahoe by adding a Fast Recovery phase and by entering Congestion Avoidance instead of Slow Start after observing the four duplicate acknowledgments. Simplifying, Fast Recovery uses duplicate acknowledgments to pace retransmissions and avoids waiting for pipe draining by restarting data transmission after receiving a certain number of acknowledgments. Reno does that because it assumes that for each duplicate acknowledgment, a certain amount of segments must have arrived at the other side. In Fast Recovery, ssthres is set to $\frac{cwind}{3}$ and cwind to $\frac{cwind}{2}$, finally it adds 1 to cwind for each acknowledge it receives. In both Tahoe and Reno, whenever an acknowledgment times out, cwind is reduced to 1 MSS.

New Reno improves on Reno by modifying the Fast Recovery phase. In particular, New Reno sends a packet from the end of the congestion window whenever it receives a duplicate acknowledgment. Once in Fast Recovery, New Reno records the highest outstanding unacknowledged packet sequence number returning to Congestion Avoidance once this sequence number is acknowledged. One systemic problem in New Reno is that it can enter Fast Recovery even if there is no packet lost when more than three packets are delivered out of order [59].

SACK modifies Fast Recovery too. In this phase, SACK keeps track of the estimated number of outstanding packets in the path in a variable called pipe. The sender retransmits data when the pipe is smaller than the cwind, this separates the decision of when to send packets from the one of which packets to send. SACK retains the memory of previous acknowledgments and uses it at retransmission time to decide whose packets to retransmit to avoid sending multiple times the same ones [60].

Vegas [58] differs from Reno in the way it detects loss and bandwidth and in the Slow Start phase. During Congestion Avoidance, Vegas estimates the amount of data buffered at switches, and based on that, it increases or decreases the value of the congestion window. This process is done by estimating the expected rate, a number Vegas obtains by keeping track of acknowledgments' RTT and assigned to cwind. During Slow Start, Vegas increases cwind every other time compared with Reno and terminates this phase upon detecting queue buildup attempting to avoid Reno's typical Slow Start losses.

Compound TCP (CTCP) designers wanted it to be efficient inside networks characterized by high bandwidth-delay products. Moreover, they also wanted the protocol to be fair towards other CTCP flows characterized by different RTT and sharing the same link. Finally, they wanted CTCP to be fair with other non-CTCP flows (this characteristic is also called TCP friendliness). Compared to previous iterations, CTCP modifies its Congestion Avoidance phase to consider both RTT and loss. While normally behaving like Vegas, upon

detecting loss CTCP adjusts the cwind in one of the following ways. If there is no congestion, cwind is grown exponentially. Otherwise, if the RTT component detects loss, the cwind is decreased proportionally to the number of backlogged packets. Finally, if it is the loss component that detects loss, cwind is drastically decreased, the active phase is changed to Congestion Avoidance, and the RTT component is disabled until CTCP goes back to Congestion Avoidance [52].

Westwood improves on New Reno's sender side to better handle large pipes characterized by a large bandwidth-delay product. In particular, Westwood performs bandwidth estimation by analyzing characteristics of the acknowledgments and uses it to set the cwind and the ssthres after a congestion episode instead of simply halving cwind [61].

### 4.3.2 CUBIC

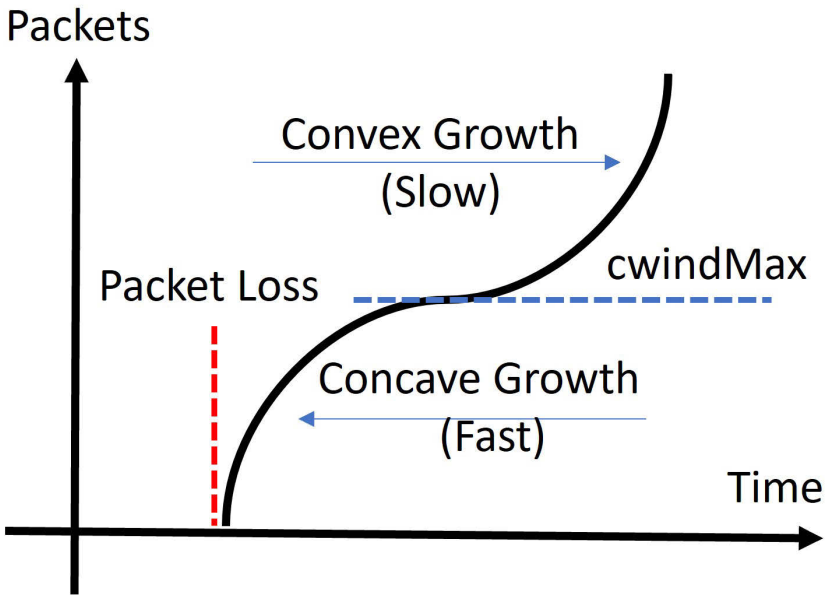



Figure 4.1: CUBIC Growth function.

TCP CUBIC is a congestion avoidance algorithm designed to overcome the low utilization problem characteristic of long-distance networks. Reno and its variants (such as New Reno) behave suboptimally in these networks because they use a linear function to make the cwind grow following a congestion event, considerably slowing the cwind growth in networks characterized by a large bandwidth-delay product. On the other hand, in TCP CUBIC, the window size is a cubic function of the time since the last congestion event, and it's mostly independent of the RTT.

Upon observing multiple duplicate acknowledgments, CUBIC memorizes the cwind as cwindMax. It then decreases the cwind and performs the Fast Recovery and Re-transmit phases of previous TCP versions. When CUBIC enters Congestion Avoidance from Fast Recovery, it increases the cwind using the concave profile (fast growth) of the cubic function set to plateau at the cwindMax. After that, the cwind grows following the convex profile (slow growth) of the cubic function.

Figure 4.1 shows the CUBIC growth function after a loss event. The cwind is shown to grow fast up until the cwindMax that was detected at the congestion point. The growth then becomes convex while CUBIC

probes for more bandwidth. Controlling the cwind in this way has been found to improve fairness between connections characterized by different RTT, to improve protocol and network stability by avoiding packet burst characteristics of purely convex driven growth functions, and to maintain higher throughput. Improvements in fairness come from the fact that in RTT based implementations, connections with smaller delays can receive acknowledgments faster and can consequently increase the window size quicker than connections characterized by longer delays.

Researchers have found several issues in CUBIC caused by the fact that it was developed for short-distance connections with extended delays. In particular, CUBIC's exponential increase that happens after the Slow Start phase can cause several segment losses [62], and by conflicting with other flows in the same state, it can quickly drain switches resources.

### 4.3.3 BBR

Bottleneck Bandwidth and Round-trip propagation time (BBR) [2] is arguably one of TCP's most sophisticated congestion control algorithms. BBR builds on top of two fundamental assumptions. The first assumption implies that the end-to-end path can be assimilated to a single link characterized by the bandwidth of the bottleneck link, and by the RTT of the end-to-end path. The second assumption is that it is possible to build accurate estimates of RTT and bandwidth for short time windows. Considering that, BBR periodically estimates the maximum available bandwidth and minimal RTT.

BBR's algorithm has four phases executed in order [63]: Startup, Drain, Probe Bandwidth, and Probe RTT.

During the Startup phase, BBR behaves similarly to CUBIC by doubling the bandwidth at each RTT until it assumes to have reached the bottleneck upon observing that the measured bandwidth does not increase anymore. Since the measurements are delayed by one RTT, a queue is formed. This queue is drained in the Drain phase by temporarily reducing the throughput. In the Probe Bandwidth phase, BBR executes numerous cycles of bandwidth increase and Queue Drain to try and increase the bandwidth if possible. After reaching a result, BBR passes to the Probe Latency phase in which it drastically reduces the throughput to 4 packets for a short period to ensure that queues are drained and improve RTT estimation. By executing these phases, BBR wants to provide stable in/out bottleneck rates and full pipe utilization. By accomplishing these two objectives, BBR can achieve good performance without queue buildups along the path.

Nonetheless, developers have observed several problems in using BBR [63]. BBR flows appear to act unfairly towards other flows characterized by different RTT. Moreover, BBR's slow adaptation rate can cause bottleneck overestimation when competing with other flows in the presence of shallow buffers, insufficient queue draining during RTT estimation, and conflict with other congestion control algorithms that can cause significant packet loss. For these reasons, BBR may underperform when applied to networks characterized by variable performance.

## 4.4 TCP Alternatives

This section introduces several important TCP alternatives. These protocols have been developed to satisfy specific needs such as handling multiple flows per connection or improving performance in degraded environ-

ments.

### 4.4.1 SCTP

SCTP [64] is a message-oriented transport protocol that was designed to optimize the transmission of multiple streams of data over a single connection and to support multi-homing by separating endpoints identification from IP addresses. During the connection phase, SCTP endpoints exchange specific tags that will identify the connection allowing communication even in case of IP change. Moreover, SCTP supports multiple flows within a single connection by delivering blocks of data in chunks. Each SCTP packet exchanged during a session is composed by a common header section and by one or more data chunks. Each data chunk can have an application tag used to differentiate between flows.

SCTP uses a TCP loss-based congestion control described in RFC 2581 [65] in which cwind is slowly increased in the absence of congestion and drastically reduced after a congestion event. SCTP performs congestion control over entire associations (as opposed to individual streams), keeping a separate cwind for each IP destination.

Chief problems of SCTP are its rather old congestion control algorithm and the fact that many middleboxes and firewalls are not designed to work with it.

### 4.4.2 UDT

UDP-based Data Transfer protocol (UDT) [66] is an application-level data transport protocol built on top of UDP and designed to transfer large volumes of data over high-speed links. UDT provides basic QoS control such as allowing full and partial message-oriented reliability, sequenced or unsequenced delivery, and time deadlines on transmission retrials. Similar to SCTP, UDT also supports multiple flows within the same connection.

By default, UDT uses periodic acknowledgments to progress transmission and negative ones to notify senders of loss. Periodic acknowledgments are particularly performant when transferring lots of data in fast networks because they reduce the number of control messages, making them grow linearly with time instead of being linked to the number of packets sent. UDT congestion control uses two mechanisms called Window and Rate Control. Window Control estimates bandwidth through packet-pair probing and by growing the cwind in a way that is inversely proportional to the available bandwidth. Instead, Sender Rate Control uses the data arrival rate at the receiver side to limit the transmission rate accordingly.

UDT also supports user-redefinition of the congestion control implementation by allowing developers to implement callbacks that give control over congestion control variables.

### 4.4.3 QUIC

QUIC is a UDP-based application-level communication protocol developed by Google in 2013 to replace TCP and to overcome a number of its limitations. To speed-up connection establishment, QUIC implements a zero RTT security and migration algorithm for connections with known servers. Similar to UDT, QUIC allows the

implementation of user-defined congestion control algorithms, provides facilities to overcome HoLB limitations by multiplexing flows, and implements forward error correction to reduce retransmissions and increase throughput [3].

QUIC can establish connections and begin sending data in 0-RTT because endpoints that have met before preserve the connection status. Similar to SCTP, the IDs that identify the connection are not related to the endpoints' IPs eliminating multi-homing problems. If the client and server have never connected before, the secure handshake takes 1 RTT to complete. 2 RTT are instead required when there is a need for negotiation or when the security keys need to be changed [3]. Similarly to SCTP, QUIC overcomes HoLB by supporting separate data streams inside a single connection. Each stream behaves like a TCP connection at the expense of some overhead. Each stream requires a specific identifier and offset to keep track of flows within a single connection.

QUIC decouples congestion control from reliability. QUIC's reliability mechanism cumulatively acknowledges the latest packet received, and similar to TCP, it uses selective acknowledgments to register the reception of loss packets without being limited to sending a maximum of three SAck blocks. In lossy environments, this approach reduces erroneous retransmissions and removes the need for timeouts [67]. QUIC acknowledgment messages also carry information concerning the time a packet was received and when the relevant acknowledgment was generated, providing to the sender important information about the connection.

QUIC does not propose a congestion control algorithm but instead provides an API to add an implementation. For example, Google uses a modified algorithm similar to TCP CUBIC that provides more information to the congestion control such as being able to differentiate between transmissions and re-transmissions by analyzing the unique incremental number that is associated with each packet.

Two final considerations are that QUIC is the only protocol that had security being part of its design and that it will be HTTP/3's transport protocol of choice [68].

### 4.4.4 Mockets

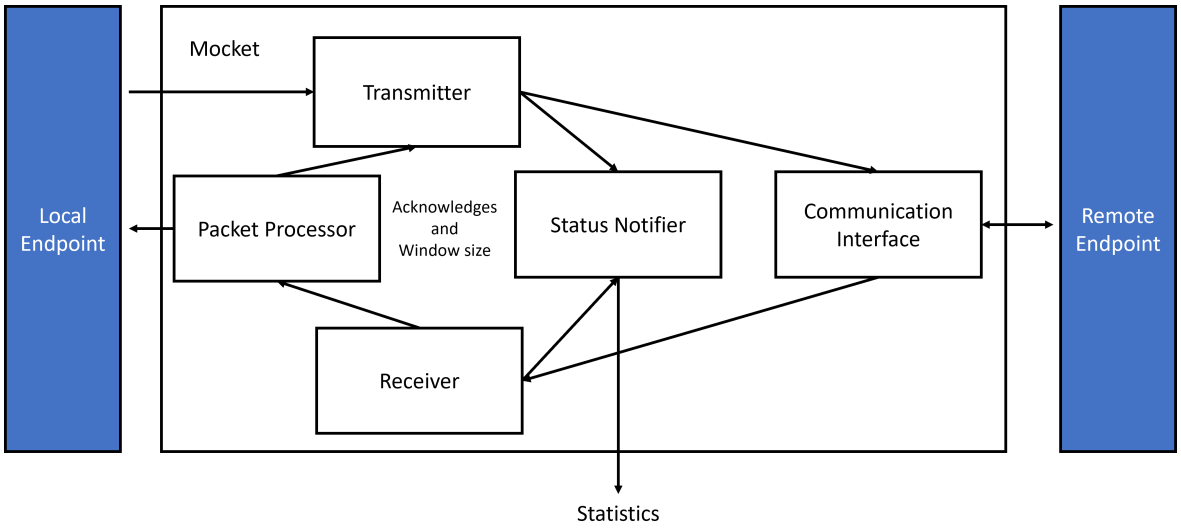



Figure 4.2: Mockets design.

The Mockets communication library [69] is an application-level data transport protocol designed to overcome challenges that characterize tactical networks developed in response to limitations observed in other protocols, TCP in particular. Mockets's API allows applications to specify different classes of services such as: reliable, unreliable, sequenced, and unsequenced. The library also supports message tagging and then provides an interface to change priority, cancel, or replace tagged data. Mockets also supports node mobility, security through the Bouncy Castle implementation of DTLS, and is equipped with a passive bandwidth estimation algorithm that is discussed in section 6.3.

Mockets also contains a condition monitoring infrastructure [15] capable of informing applications with timely and accurate information about various characteristics of the connection between two endpoints. Mockets can detect peer unreachability through a keep-alive mechanism, and applications can register callbacks to be notified once a configurable period has passed without the local endpoint receiving any message from the remote one. Mockets also harvests several other metrics such as the number of retransmissions, duplicated, sent, and received packets and bytes per service class. Finally, Mockets can also perform latency estimation through acknowledge- and timestamp-based algorithms. The acknowledge-based algorithm measures the RTT as the difference seen by the sender between sent packets and their acknowledgments. The timestamp algorithm operates by periodically adding a timestamp chunk to packets that do not fill the Maximum Transmission Unit (MTU). Mockets's implementation accounts for processing time in this measure. While this second algorithm is generally more accurate, numerous conditions could cause these samples to over or under-estimate the RTT, considering that, Mockets merges and smooths the two types by implementing an exponential weighted moving average.

Figure 4.2 shows Mockets internal design. There are five major components called Transmitter, Packet Processor, Receiver, Status Notifier, and Communication Interface. The Communication Interface can either be a UDP socket or be specified by an application. Developers can use this feature to extend Mockets's transport layer behavior by implementing specific functionalities or adding support for other socket implementations. For example, developers could prevent Mockets from handling reliability and sequencing and implement custom logic inside the Communication Interface. The Transmitter's primary tasks are to prepare and send messages. This component also monitors the status of the queues used to manage the different classes of service. Finally, it supports several other features such as bandwidth estimation, latency estimation, and transmit rate modulation.

The Receiver is a layer between the Communication interface and the Packet Processor used to separate the receive from the handling logic. The Packet Processor analyses and puts together buffers passed by the Receiver and then notifies the application once a message is ready. The Packet Processor also notifies the Transmitter about received acknowledgments and duplicate packets. The first information is used to update the window size and the second to re-schedule acknowledgments that may have not reached the other endpoint. The Transmitter and Receiver update the Status Notifier with information about the connection state which then delivers this information in protobuf encoded UDP messages.

The local endpoint uses Mockets's API to accept, connect, send, and receive packets, with a syntax similar to TCP or UDP sockets. An application can also provide extra parameters to specify the class of service of a message or tag it with an ID which can be used in conjunctions with other functions of the API to cancel or replace one or more messages that share the tag.

Once an application passes a message to Mockets, the library verifies that the size is smaller than the MTU or otherwise divides it into smaller fragments before passing it to the Transmitter. Each message is then stored in a specific queue accordingly to its QoS requirements. A transmit rate modulation algorithm is used to provide pacing and congestion control.

Mockets connections can be secured using Bouncy Castle's DTLS implementation [70]. This works by first establishing a Mockets connection and then securing it with DTLS on top of it. Both Mockets and DTLS support the definition of connection unique IDs to avoid having to perform a handshake each time a network handover or a reconnection is necessary.

## 4.5 Experiments

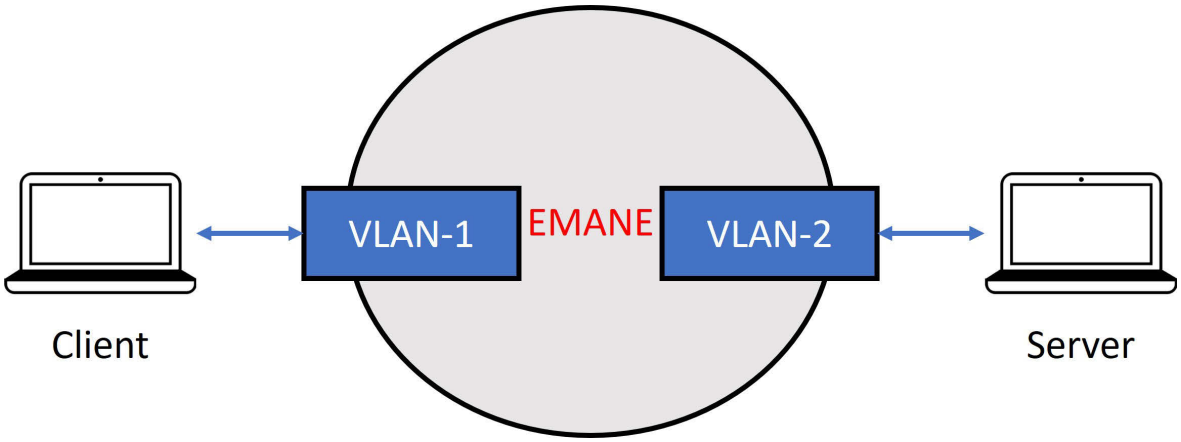



Figure 4.3: Testbed configuration used for the comparison of unicast protocols.

To evaluate the progress of COTS transport protocols when applied to tactical environments, we selected several promising solutions and performed multiple experiments representative of today's and tomorrow's needs in a realistic tactical scenario. In particular, we evaluated the performance of the following communication patterns: handshake, Remote Procedure Call (RPC), unidirectional transfer, and bidirectional transfer. We run experiments using TCP, SCTP, UDT, QUIC, and Mockets. For TCP, We decided to analyze three stable and refined congestion control implementations: CUBIC, CTCP, and BBR.

We decided to analyze handshake performance because it is a necessary operation and because multi-step handshake mechanisms can significantly degrade or impede communication in the presence of slow and lossy networks. We analyzed asymmetrical RPC like communication because this pattern is at the basis of communication with web services and REST-based interfaces that are becoming increasingly more common. Finally, we examined one-way and two-way file transfer because they can give insight into the steady-state performance of each protocol. Moreover, since we knew the amount of data to transmit, we could quantify control overhead. Finally, we used the two-way test to measure the effect of competing traffic flows on acknowledgment mechanisms since it can greatly influence transmission speed.

For the test, we used a network environment based on Extendable Mobile Ad-hoc Network Emulator (EMANE), a sophisticated framework for real-time modeling of mobile network systems [71]. In particular, we used EMANE to dynamically control network characteristics such as latency, packet loss, and network bandwidth

of traffic that flowed through controlled VLANS.

### 4.5.1 Testbed Notes

| Protocol | Implementation |
|---|---|
| SCTP | Linux Kernel Stream Control Transmission Protocol Tools v.1.0.16 |
| UDT | UDT-4 as provided by libudt-dev package |
| QUIC | ngtcp2 library |

Table 4.1: Protocol versioning when different from what available in Ubuntu 16.04.

| Test | Measurement | Transmission Size |
|---|---|---|
| Handshake | Time measured before and after connect event. | N/A |
| RPC | Time measured before the request transmission from the client and after complete reception by the server | Request of 256B and Responses of 2048B |
| One-Way transmission | Time measured between first and last byte as received by the client | 1MB |
| Two-Way transmission | Time measured by the client between connection establishment and complete file reception. | 1MB |

Table 4.2: Summary of time measurements for each test.

To drive the experiments, we developed a simple C++ test harness that we could use to specify a communication pattern and a protocol to be tested. We then deployed the test harness in two endpoints, a client and a server, each connected to a specific VLAN. We emulated the connection between the two VLANs using EMANE. Figure 4.3 summarizes this configuration. Each endpoint was deployed in a Ubuntu 16.04 (kernel 4.15) virtual machine. Table 4.1 summarizes the version of protocols that were not already available in Ubuntu. In each test, we measured how long it took to complete a task from the perspective of the client application.

1. In the handshake test, we measured the time before and after the connect call;
2. In the RPC test we measured the time before and after the RPC (performed by the client);
3. In the one-way data transmission we measured the time from the receiving of the first and last byte sent by the server;
4. In the two-way data transfer we measured the time between the connection establishment and the complete file reception.

During the RPC test, client and server exchange 256B for each request and 2048B for each reply. During the file transfer tests, the file size was 1MB. The previous information is summarized in Table 4.2.

In the tests, we focused on evaluating the link between performance and packet loss. We selected a representative one-way bandwidth of 512Kbps and a link latency of 250ms. We kept that fixed and then executed four experiments each with a different average packet loss ranging from 0% to 15% with a 5% increment. We then executed each of these four configurations ten times per test per protocol for a total of 1120 runs.

### 4.5.2 Results

Figures 4.4 to 4.7, summarize the results of the experiments. From the figures, it is possible to observe a general negative trend in performance at the increase of packet loss. A lack of results for some of the tests, especially ones with high levels of packet loss, can be attributed to repeated failures or excessive time to complete.

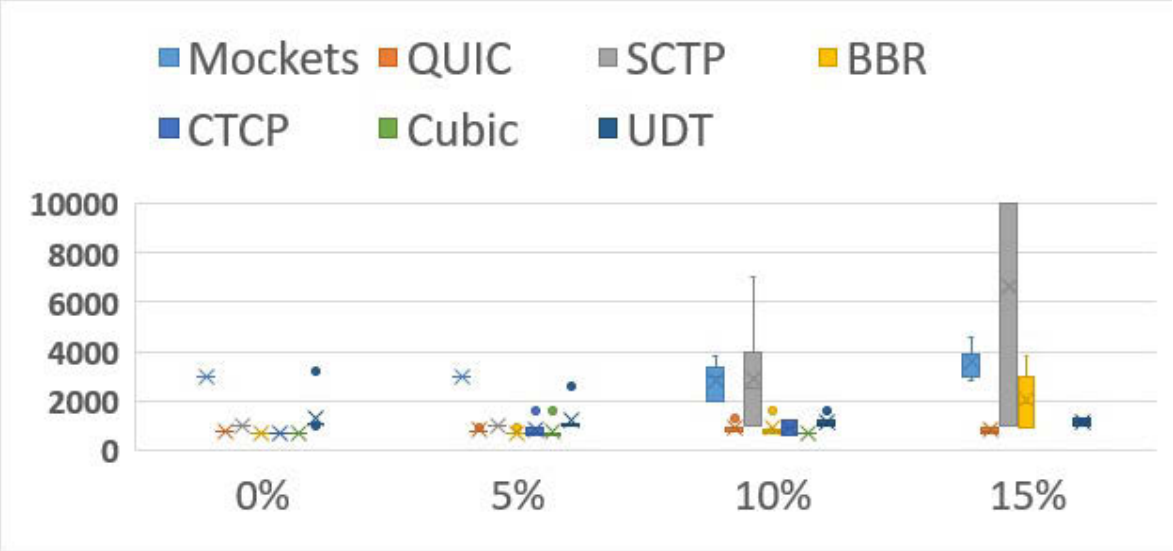



Figure 4.4: Distribution of handshake times under four different packet loss configurations.

Figure 4.4 aggregates the results of the handshake experiments showing that as the packet loss increases, all configurations take longer to complete. This result is likely caused by the fact that all protocols wait a certain amount of time before retransmitting handshake messages thus lengthening the process.

Figure 4.5 shows the result of the RPC experiment. In this case, the figure shows an average increase in the time needed to complete the procedure. It is worth noting that QUIC handled this experiment worse than the handshake one and that there may be value in investigating if there is any difference in the way reliability is handled before and after completing the handshake phase.

BBR showed the wider variability during the tests with 10% packet loss, and it was the slowest protocol during the 15% test. Mockets obtained much better performance after the handshake. This is likely caused by the fact that Mockets handshake is based on static timers that must be opportunely tuned to provide quicker handshakes in networks characterized by packet loss and low latencies.

Figure 4.6 shows the average throughput achieved in the one-way file transfer experiment over the different packet loss configurations. In this experiment, CUBIC showed a very low throughput. Since all packet loss

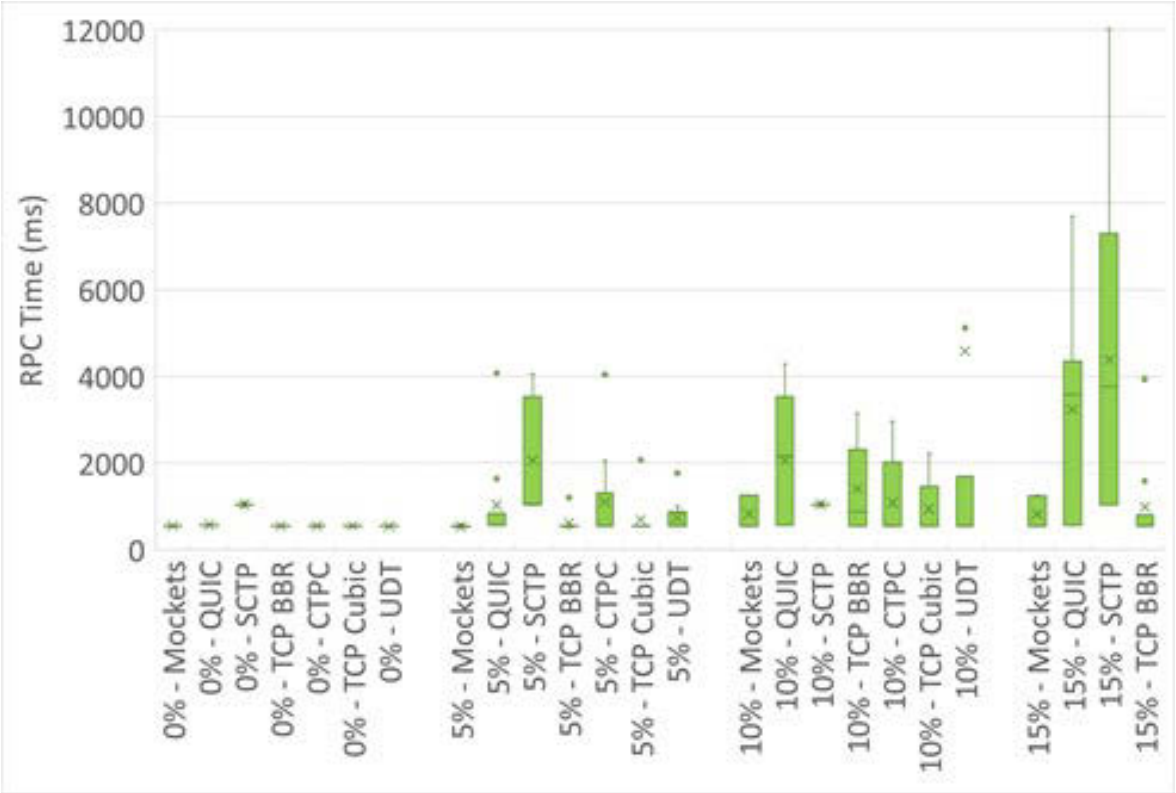



Figure 4.5: Distribution of RPC times under four different packet loss configurations.

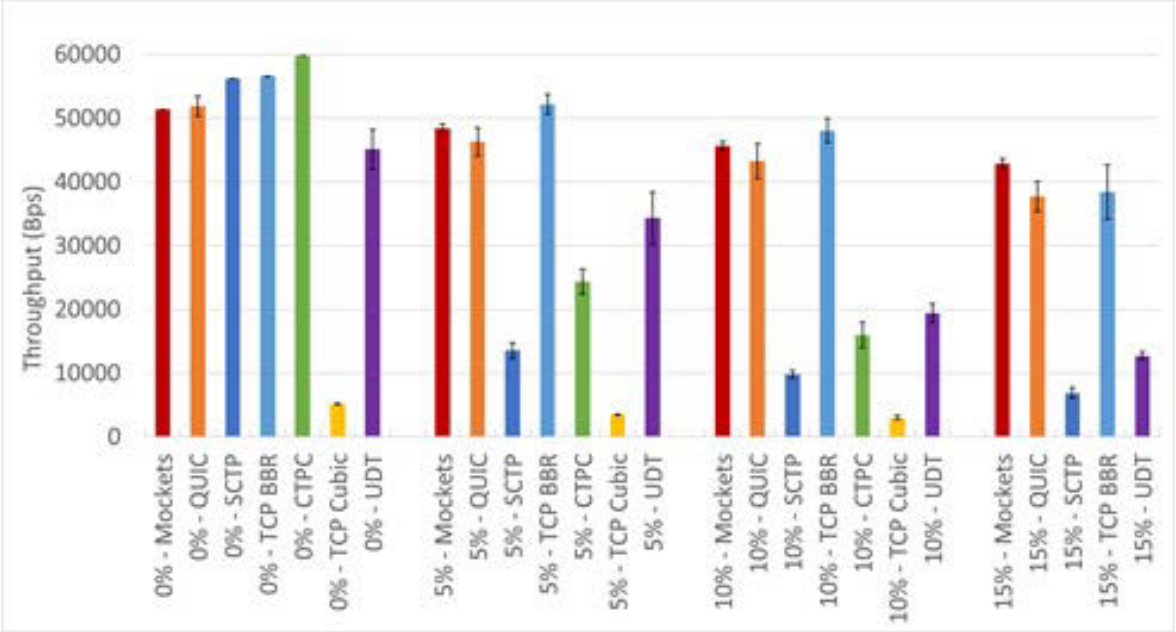



Figure 4.6: Average throughput during the one-way file transfer experiment.

configurations share this result, the cause likely has to be found in the protocol's inability to handle low bandwidth when paired with high latencies.

CTCP shows excellent throughput when packet loss is absent, but performance degrades quickly in the other configurations. The one-way file transfer experiment shows that Mockets can achieve higher throughput than other solutions as the packet loss increases. It is also likely that the cause of bad performance in the lower packet loss configurations has to be attributed to a default tuning more suited to degraded networks than to

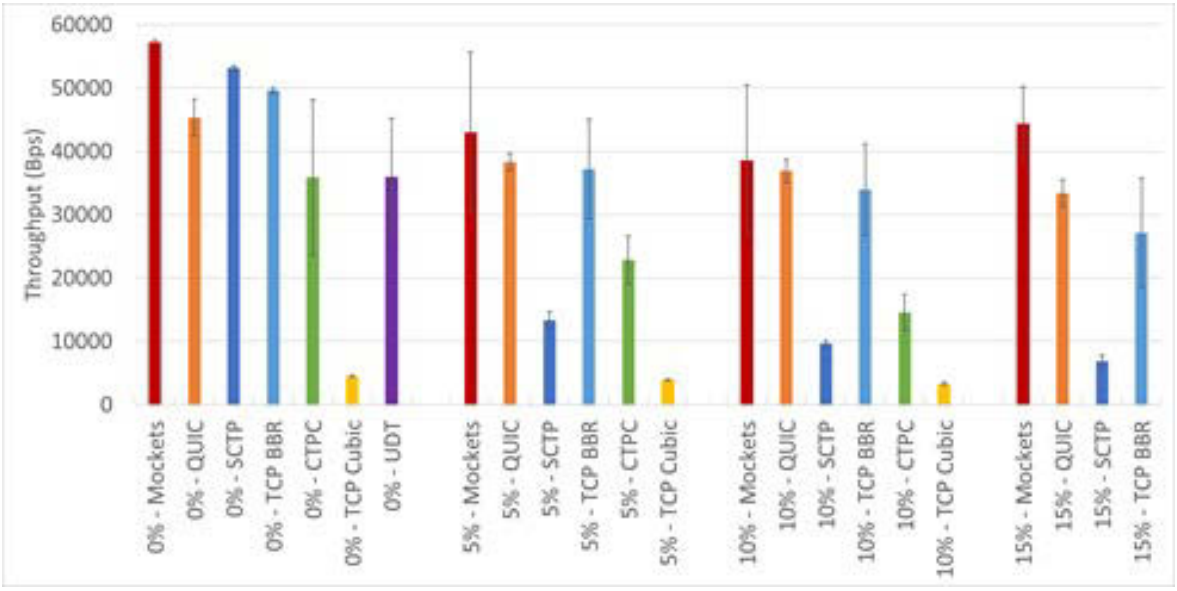



Figure 4.7: Average throughput during the two-way file transfer experiment.

resource-rich and reliable ones. BBR and QUIC also achieve excellent performance in the presence of loss.

Finally, Figure 4.7 summarizes the results of the two-way file transfer experiment. Mockets and QUIC have the best performance. BBR and SCTP also achieve good performance. The difference in results is most likely caused by how certain protocols can better take care of transmitted packets to relay acknowledgments (Mockets, for example, can piggyback them in data packets) and the negative impact that the reverse flow has on the reliability of acknowledgments.

# Chapter 5

# Experimental Evaluation of Group Communication Solutions in Constrained Environments

Group communication can describe a variety of communication modalities and semantics that involve many-to-many communication in which participants, instead of designating recipients, transmit messages addressed to a group or topic. Nodes then task the group communication solution to filter and deliver only information that satisfies their interest.

This chapter first introduces IP-Multicast, a communication protocol that forward messages sent towards a specific multicast address to any node that joined the multicast group identified by that same address. Since IP-Multicast is generally a best-effort protocol, it is not natively suitable for many applications that require higher levels of guarantees on the reliability of the communication channel or the preservation of the arrival order of messages. Luckily, the need for timely and reliable information delivery between groups of nodes is not peculiar to TDR operations, and many other commercial drivers such as the Internet of Things, Healthcare, Smart Cities, Monitoring, and the emergence of Fog/Edge computing and Information-centric Networking have led to the development of an ever-increasing number of solutions capable of providing reliable and ordered delivery. In particular, the Publish/Subscribe architectural pattern has seen significant adoption and become a common design pillar.

Group communication solutions separate into two categories, decentralized and centralized approaches. Decentralized systems use multicast or broadcast, while centralized ones use one or more federated brokers that connect individually to clients. On rare occasions, solutions realize combinations of these two approaches. Designed for the commercial Internet, COTS group communication solutions present the same classes of concerns expressed in previous chapters when deployed in significantly bandwidth-constrained environments that exhibit variable latency and significant node mobility. This fact begs several questions that ought to be answered. Can these solutions efficiently support communication in TDR operations? How can they be evaluated in a realistic environment? To answer these questions, we performed extensive research evaluating several group communications protocols for data dissemination and extracted some of their most characterizing features. Moreover,

we evaluated these solutions over an emulated tactical edge network using the Anglova Scenario, a realistic military scenario that describes nodes' movement patterns and connectivity over time, EMANE, a sophisticated emulation framework, and a test-harness specifically designed to ensure consistency in collecting and analyzing results.

The rest of this chapter is as follows. The first section introduces IP-Multicast and explains why this communication protocol is not sufficient on its own to provide stable and reliable group communication. The second section discusses the Publish/Subscribe architectural paradigm as a fundamental component of modern software infrastructure to provide many-to-many communication. The third section contains a survey of group communication characteristics that can be useful to understand group communication protocols. The fourth section presents all the protocols examined in the experiments. The fifth, sixth, and seventh sections introduce the various components of our realistic experimentation environment. Finally, the remaining four sections present a variety of experiments.

## 5.1 Multicast

IP-Multicast [72] is a simple technique to implement one-to-many and many-to-many communication over an IP infrastructure [73]. A group of hosts can perform a multicast communication by agreeing on a multicast group address and a port. Upon sending a message towards that address, all the hosts listening to it will receive the message. A multicast group address is an IP address that belongs to the multicast range from 224.0.0.0 to 239.255.255.255. To participate in a multicast group, receivers will need to inform the network of their interest in messages sent towards a specific multicast group address by sending an IGMP [74] join packet to their closest router. The network box will then forward multicast packets to the local receivers. Upon receiving multicast messages, network nodes such as routers and switches will also replicate and share messages between them and to interested neighbors as necessary.

Many characteristics make IP-Multicast an interesting choice to provide group communication. To start, IP-Multicast does not require participants to have any prior knowledge of the number of receivers or publishers. Furthermore, IP-Multicast is intrinsically more efficient than unicast for group communication because even if a packet has to be sent to multiple receivers, it requires the sender to send it only once. Another advantage is that routers do not need to keep track of routes towards each single multicast destination but can instead forward messages over tree-like data structures built based on whether or not any receiver or router manifested interest in a multicast group address. IP-Multicast also comes with some downsides. For once, while nodes belonging to the same local network can generally use multicast, the same is not necessarily true over wide networks, which often require network administrators to perform specific configurations to support multicast traffic.

While IP-Multicast does not force a specific transport protocol, it is commonly associated with UDP, and consequently, shares the same best effort guarantees that characterize datagram transmission. It follows that IP-Multicast does not address many other typical required functionalities such as the need for granular control over messages QoS, reliability, ordering, and security. Moreover, due to the peculiar nature of radio channels, which are lossy and not constrained by a cable medium, many wireless radios implement their custom handling

of multicast traffic to circumvent performance degradation, making their performance unpredictable over heterogeneous solutions. Finally, routers and network boxes often do not forward multicast traffic due to reverse path forwarding failures [75], security concerns, and Time-to-Live limitations.

## 5.2 The Publish Subscribe Architectural Paradigm

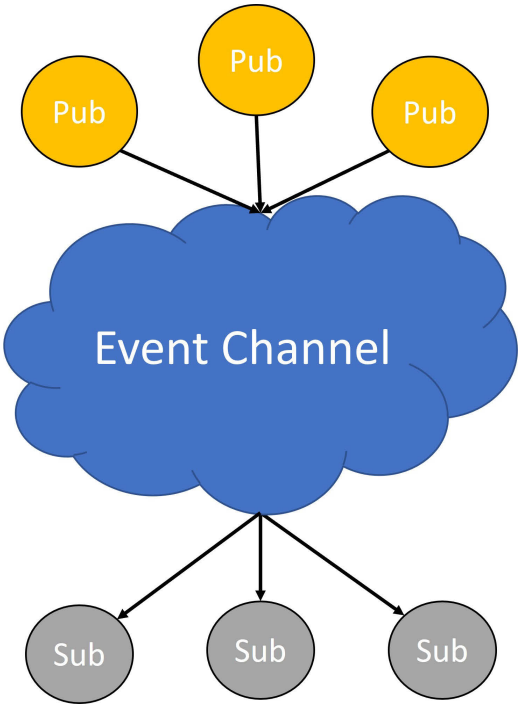



Figure 5.1: Publish Subscribe architectural pattern.

Produced by the need for a better solution for group communication than IP-Multicast, the Publish/Subscribe paradigm is a state-of-the-art approach to satisfy the need for accessible loosely-coupled asynchronous group communication between distributed components.

The Publish/Subscribe architectural pattern describes a communication system in which anonymous, asynchronous, many-to-many, loosely-coupled nodes transmit notifications to interested receivers. Figure 5.1 summarizes this architecture and shows the three typical elements that compose a Publish/Subscribe solution: publishers, subscribers, and the event channel. The publishers insert information into the system, the subscribers receive the information they are interested in, and the event or communication channel is an abstraction over the medium used to signal interest and move data between nodes. The event channel can be implemented in a centralized or decentralized fashion depending on whether or not it assumes the form of a centralized unit (generally called a broker) or of a distributed network of nodes that share messages between each other.

On top of satisfying basic communication needs, it is common for Publish/Subscribe implementations to also provide support for a vast array of highly valuable secondary features, such as mechanisms to guarantee message reliability, ordering, delivery deadlines, filtering, security, persistency, connection health, group management, and delivery synchronization.

## 5.3 A survey on group communication characteristics

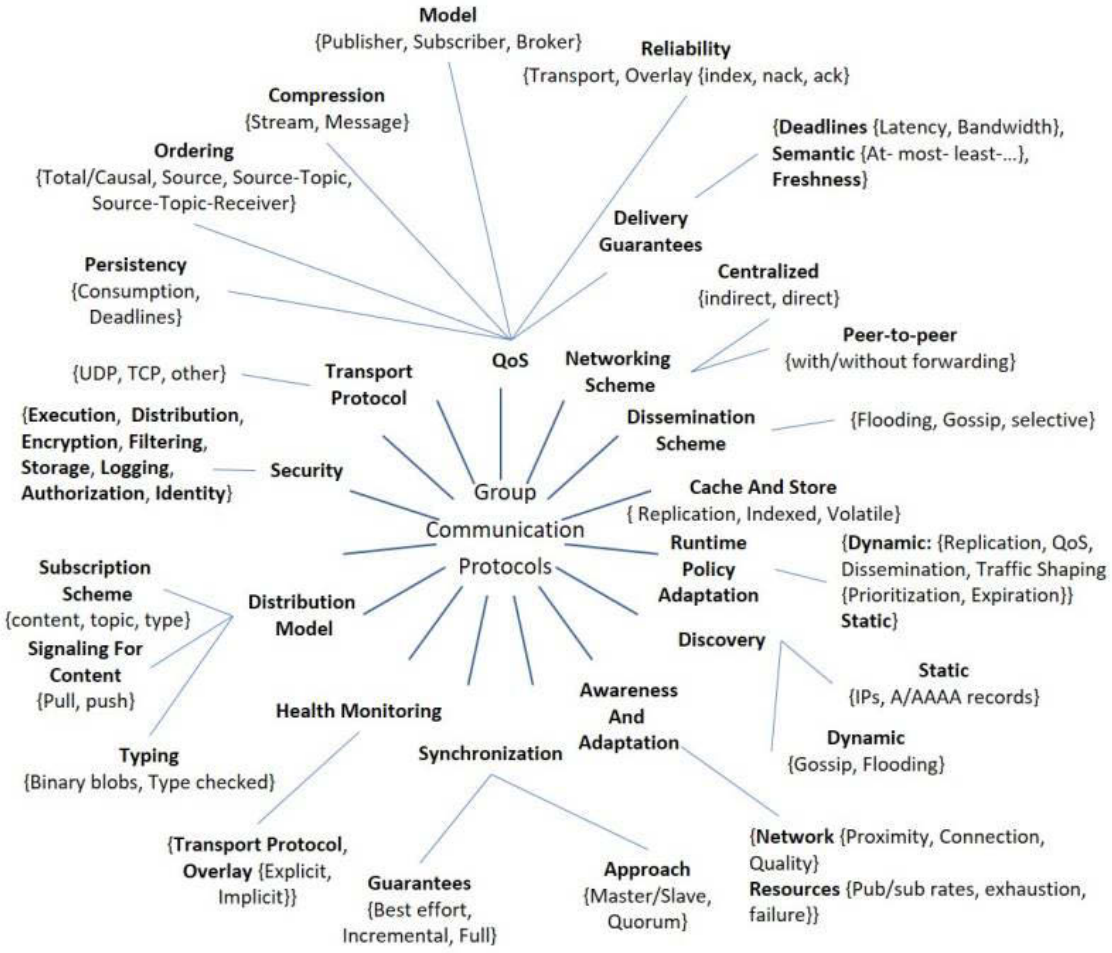



Figure 5.2: Important characteristics of group communication solutions.

This section contains an analysis of fundamental characteristics of group communication solutions with an eye to TDR operations. The list of analyzed features is summarized in Figure 5.2.

### 5.3.1 Networking Scheme

The Networking Scheme describes the structure of the event channel. To illustrate this we consider three characterizing examples: Direct-Centralized, Indirect-Centralized, Decentralized. In Direct-Centralized solutions, a centralized entity (typically called a broker) receives messages generated by publishers and delivers them to interested subscribers. A group of nodes associated with a single primary broker is generally called a cluster. Over large networks, clusters are connected by federating the brokers. In Indirect-Centralized solutions, publishers and subscribers exchange events directly but require a centralized entity to sustain the event channel, discover each other, or find data. Conversely, the event channel in Decentralized solutions is made up of publishers and subscribers themselves. Decentralized solutions may be static or dynamic depending on whether or not the topology can change at run-time.

Centralized solutions are arguably simpler to implement and maintain, require less configuration, and provide a smaller security attack surface by concentrating their capabilities in a single entity. On the other hand, distributed solutions can better handle, at least in theory, the disjointed nature of the battlefield. Nodes operating

in ad hoc networks may abruptly change their location and lose contact with the centralized entity. In these scenarios, distributed solutions capable of relaying messages achieve wider coverage, reducing the impact of mobility on the overall communication reliability. Decentralized solutions are also intrinsically less vulnerable to denial of service attacks.

Another principal limitation of centralized solutions is that the broker position limits their forwarding capabilities. The fact that only nodes reachable by the broker can receive and forward messages, creates the need of selecting a suitable position for the broker in terms of its ability to reach all members of the cluster and possibly brokers of other clusters. In the absence of routing, brokers may have to be deployed into gateway nodes capable of reaching the back-haul connection, but this may reveal to be sub-optimal for communication within the cluster.

### 5.3.2 Transport Protocol

Communication middleware and other solutions designed to tackle group communication have the advantage of abstracting and simplifying network interaction. Nonetheless, the choice of transport protocol is strongly linked to performance, especially in TDR operations as discussed in Chapter 4. Group communication solutions are generally based on TCP, UDP (unicast or multicast), or other ad hoc protocols built on top of UDP. Considering the observations produced in Section 5.1, multicast is a more attractive option in terms of bandwidth consumption, but it is also more complicated to adopt and may not work in wide networks.

### 5.3.3 Distribution Model

The Distribution Model describes the process that allows a node to subscribe to specific data, which we call Subscription Scheme, and the method used to trigger data distribution, which we call Distribution Scheme. Common Subscription Schemes are topic-, type-, and content-based, while examples of Distribution Schemes are pull- (or fetch) and push-based.

Topic-based subscription filters events based on unique identifiers associated with each event called topic. Topics can generally support hierarchical organizations to enable granular sub-specifications while keeping APIs simple. For example, if a node wants to subscribe to information generated by a weather sensor deployed in a tower called x, the subscription topic could look like ”tower.x.weather”. These APIs typically support wildcards to extend their functionalities, such as the ”*” symbol for general substitution and the ”>” symbol for hierarchical substitution. For example, ”tower.*.weather” could be used to retrieve weather information from all the available towers, and ”tower.*.>” could be used for all types of information.

Type-based subscriptions use information types to filter information. This approach generally requires defining all the types of messages that publishers and subscribers will use at run-time.

In content-based subscriptions, subscribers advertise their interest by setting specific triggers on events’ informative content typically expressed as key-value pairs. The triggers can be meta-data associated with each event or data fields extracted by inspecting events’ content. Depending on the implementation, content-based subscriptions may too require all types to be defined beforehand.

The subscription mechanism holds consequences both in terms of usability and performance. Topic-based sub-

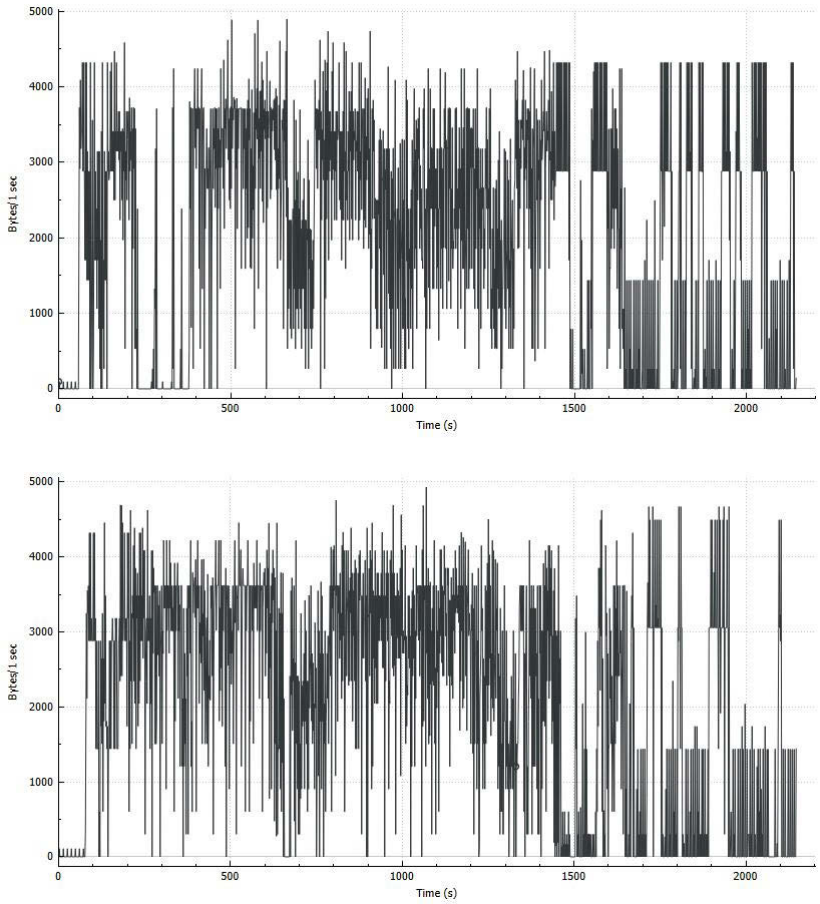



Figure 5.3: Comparison between throughput obtained with short (top) and big (bottom) topics.

scriptions are flexible since they can associate arbitrary blobs of data and topics, but they introduce significant overhead when lengthy. Type and content-based subscriptions are generally more efficient because they intrinsically contain the information that the event channel will use for matching, but require preliminary knowledge of what will be transmitted, reducing flexibility and increasing development complexity. For example, Figure 5.3 shows the comparison throughput of the same communication middleware using big and small topics made respectively of one and ten characters. On average, the small-topic configuration used 25 fewer bytes per packet.

For what concerns the Distribution Scheme, pull-based approaches have the consumer request information, typically by sending fetch messages, and push-based solutions have the publishers send data to the consumers whenever it matches their interests. The Distribution Model also describes if a solution transmits blobs of binary data or if data types need to be specified to all participants before they can be shared and the maximum and recommended size of messages exchanged.

Theoretically, pull approaches should achieve better performance in TDR networks because they are more conservative in how they use network resources by only requesting data they are ready to process. On the contrary, high latencies may be more favorable to push mechanisms because they are not affected by the dely that fetch requests need to reach the event system. Sophisticate algorithms can mitigate this problem by estimating the best time to send fetch requests based on consumer performance.

### 5.3.4 Dissemination Scheme

We use the term Dissemination Scheme to classify Group Communication Solution (GCS) on the approach they use to distribute messages between clusters of nodes. Common approaches are flooding, selective, and gossiping [76].

In flooding, each node broadcasts received messages to any node it can reach. In gossiping, each time a node receives a message, it forwards it to one or more neighbors it chooses randomly, de facto implementing probabilistic flooding. We call gossiping ”informed” if the delivery attempt of a message is guaranteed for neighbors that manifested interest to the class of that message. Finally, selection-based approaches require the static definition of information paths.

### 5.3.5 Discovery Mechanism

One of the primary challenges in TDR operations is that nodes may frequently change position, causing the communication environment to separate unexpectedly into compartmented, unreachable enclaves for long periods. A discovery mechanism is then necessary to advertise nodes' return and departure to their neighbors.

The Discovery Mechanism feature classifies solutions based on the algorithm that publishers, subscribers, and clusters, use to find and discover each other. Approaches can be static or dynamic. Static approaches rely on pre-defined lists of peers' IP addresses or DNS records, while dynamic ones are based on run-time discovery generally implemented through flooding or gossiping.

### 5.3.6 Health Monitoring

Health Monitoring describes the mechanism used to evaluate connectivity with other nodes in terms of their ability to reach each other and on the characteristics of their communication channel. Typically solutions rely on the status of the transport protocol connection or implement an overlay mechanism that exchanges extra control messages to verify connectivity and status.

We say that overlay solutions are explicit or implicit depending on whether the control messages are embedded in the event channel's messages or sent separately. Explicit solutions can implement ping/pong algorithms that judge how quickly a pong follows a ping and use that information to estimate the status of the connection. Implicit approaches use information intrinsic in the message distribution mechanism, such as measuring the difference between publishing and consuming rates. When the difference between these two grows over a certain threshold, the health monitoring mechanism can infer the presence of a problem with the node under examination.

### 5.3.7 Quality Of Service

In QoS we grouped the various mechanisms that provide control over the characteristics of message dissemination. In particular, we classified protocols based on Model, Reliability, Ordering, Delivery Guarantees, Fragmentation, and Compression.

We use Control Model to classify solutions based on who can specify QoS requirements between the publisher,

the subscriber, or the event channel. Note that a combination of these is also possible. This aspect is relevant in TDR networks since they are asymmetrical communication environments with different necessities. For example, a node closer to the headquarters may push out a lot of data, but a node at the edge may need to limit the amount other nodes can send to it to avoid network collapse. With Reliability, we classify solutions on how they ensure message delivery. A basic approach can rely on the guarantees offered by a transport protocol, more sophisticated solutions can implement separate ack/nack-based mechanisms, or use index-based fetches that associate each index to a type of information. Subscribers can then control information flow in this way by notifying the publisher (or the broker) about the index of the message they want to receive next.

With Ordering, we classify solutions based on their limits in guarantees to message arrival orders. In Total ordering, messages are delivered in absolute chronological order, i.e., no node receives a message before all other participants have received the one sent before it. Partial ordering adds caveats that relax total ordering. Partial Source-based ordering guarantees that messages sent by the same source are delivered in order, while Source-topic ordering only guarantees ordered delivery between messages sent by the same source and characterized by the same topic. It is worth noting that more restrictive classes of service require more network resources and could not be sustainable in a degraded environment.

Delivery Guarantees describe what happens if the transport protocol fails to transmit a message. Fire-and-forget approaches only provide the guarantees provided by the transport protocol, at least once, at most once, and exactly once, respectively guarantee that a message will be received at least, at most, and exactly once. Several other variations are possible, such as offering reliability but enforcing freshness by only sending the last message in a channel if previous information has not been delivered.

Persistency describes how the messaging system handles information survival to node crash, network disconnection, and time. Some options are fire-and-forget, transaction-based, time-, size-, count-, and deadline-based. Fire-and-Forget describes an approach without memory or delivery guarantees. Transaction-based maintains information until explicitly deleted. Deadline-based rules support the definition of thresholds in terms of time, size, or count, that the event channel uses to drop messages. Some examples of Deadline-based rules are to retain messages for 1 minute, to at most use 100MB for a single topic, or to remove a message only if it has been acknowledged by 5 subscribers.

Compression can be stream or message-oriented. Stream-oriented solutions are more efficient the more data is transmitted but require heuristics to decide how long to wait before sending information, complicating the solution design.

In general restrictive QoS requirements may require considerable bandwidth or significantly degrade overall performance by having well-connected nodes wait on peripheral ones.

### 5.3.8 Cache and Store

We use Cache and Store to describe how solutions preserve information between generation and delivery. Caching approaches can provide replication by proactively disseminating data to other nodes. In this case, the death of the original publisher does not prevent interested subscribers from receiving the cached data. Another approach is to share the location (an index) of the information so that interested nodes can fetch it from there

at their convenience.

We use Storing to specify if storage is volatile, when stored on RAM, or persistent, when stored on disk. RAM approaches are generally faster, but on the other hand, they are subjected to hard memory limits, and information may be lost in case of system failure.

### 5.3.9 Synchronization

Synchronization describes how the event system enforces event reception synchronization between subscribers. Examples of synchronization guarantees are best-effort, incremental, and virtual. Best-effort approaches do not provide any synchronization, with each subscriber receiving events independently from others. Incremental approaches create temporary partitions that are eventually merged with through an opportune strategy once the reason for the partition disappears. Full Synchronization guarantees that messages reach all users in identical order despite failure, however, this is generally implemented in practice as virtual Synchronization by relaxing some of the requirements while maintaining the constrain that messages reach users in identical order [77].

Another aspect worth evaluating is how to merge conflicting information. In Master/slave approaches, the state is managed by a centralized authority, while in Quorum-based ones, the state is built by having the participants vote. TDR environments are not suited for complete synchronization since parts of the networks may be partitioned for long periods.

### 5.3.10 Awareness And Adaptation

Awareness and Adaptation (A&A) describes how protocols detect and adapt to state changes such as variation in network performance and available resources. Solutions can accomplish Network A&A by shaping nodes' communication based on proximity to other nodes or subscription patterns, by implementing strategies to react to brokers' and peers' failures such as exhaustion and Publish/Subscribe rate variations.

GCSs can perform Proximity optimization by dynamically adapting information flows, making nodes obtain data from the closest source. A protocol capable of connection adaptation could detect the inability to reach a broker and switch to a different one. Quality adaptation can entail changing (publish) rates to avoid link congestion or picking a different path to distribute information. Adaptation in terms of resources could be implemented by distributing the workload between different brokers/peers to avoid exhaustion or excessive asymmetrical resources usage.

### 5.3.11 Run-time Policy Adaptation

Run-time Policy Adaptation describes the control that can be exerted at run-time. Static solutions force systems to restart after each change in semantic or other characteristics, tearing down already established channels with consequent non-negligible network overhead. Dynamic GCSs may support hot changes in QoS requirements (for example, from at most once to at least once semantic), traffic shaping (cancel or prioritize message waiting to be sent), and dissemination (changing publishing rates and information paths).

| Attack Surface | Vulnerability |
|---|---|
| Publisher | • Data Spam<br>• Corruption<br>• Repudiation<br>• Spoofed Publisher |
| Subscriber | • Subscription Spam<br>• Subscription Churn<br>• Eavesdropping |
| Event Distribution Layer | • Eavesdropping<br>• Subscription/Publishing Leak<br>• Data Inference<br>• Data corruption<br>• Malicious routing/loss<br>• Flooding<br>• Encryption Breach |
| Communication Channel | • Overlay Scanning<br>• Eavesdropping |

Table 5.1: Summary of Publish/Subscribe attack vectors and vulnerabilities.

| Security Vector | Description |
|---|---|
| Identification | How peers identity is verified and secured. |
| Authorization | How access to the system is regulated. |
| Filtering | How information access can be restricted to subsets of peers. |
| Secure Storage | How information is secured when stored |
| Secure Communication | How information is secured when in transit |
| Routing Control | How the path of information is controlled |
| Execution Control | How peers actions such as limiting the number of published messages or active subscriptions can be restricted |
| Logging and Monitoring | How monitor information is secured from abuse and eavesdropping |

Table 5.2: Summary of security points.

### 5.3.12 Security

Table 5.1 summarizes common attacks towards group communication systems [21]. One way to analyze this is to consider that there are fundamentally 4 points of entry for attackers: publishers, subscribers, the event channel, and the communication channel. For example, malicious or compromised publishers may try to spam

data generation with the objective of overloading brokers or subscribers, sending corrupted or misleading information, or being used as an intermediary to send information from unauthorized actors. Malicious subscribers may drain resources by over-subscribing, by acting in a way that causes high subscription churn rates, and by trying to eavesdrop on information to which they should not have access or interest.

If parts of the Event Channel were to be compromised, malicious brokers could try to eavesdrop on information, leak subscribers and publisher characteristics and interests, try to perform data inference and corruption, or maliciously route or lose information to compromise or delay critical communication.

Finally, external actors can attack the communication channel itself, for example, by scanning the overlay to gain information about publishers and subscribers or the data that is transmitted between them. In general, this type of threat calls for solutions that can provide access control, identity management, encrypted communication, event routing control, storage security, logging and monitoring security, and execution control (as summarized in Table 5.2).

## 5.4 Protocols

This section describes the protocols analyzed in the subsequent experiments sections of this chapter. The analysis roughly follows the same structure for each protocol. First, we provide a general introduction to the protocol, specifying the programming language it was written in and the domain it was designed for. Next, we analyze the subscription and distribution models. After that, we describe the event channel and the mechanism that the solution uses to monitor its status. Finally, we discuss how the protocol manages large networks (discussing federation and forwarding) and conclude with notes about security.

### 5.4.1 NATS

NATS [78] is a TCP-based messaging system written in Golang implementing a simple text-based publish-subscribe distribution model.

Clients connect to a NATS server through a regular TCP connection and then subscribe or publish messages in channels identified by unique topics. NATS provides a fire-and-forget and at-most-once delivery guarantee. Consequently, a client will keep receiving messages as long as its connection is active, but it will not receive messages sent before that was the case.

NATS clients monitor the connection with the server by periodically sending ping messages. If servers and clients do not exchange traffic, the server must reply with a pong message, or the clients will consider the connection stale and disconnect.

Servers can cluster and dynamically exchange information and divide clients into work queues to optimize throughput. Clusters are self-healing in the sense that once brokers discover each other, this information is shared with the clients and with other brokers so that in case of failure they can reorganize. This feature requires routing between NATS servers and between groups of clients and their respective servers. Messages can be forwarded between clusters using an internal routing protocol that allows for multi-hop configurations. Finally, NATS brokers support configurable security enclaves. They can configure authorization and authentication on

a per topic basis cluster by cluster. Specific rules can be set for the intracluster forwarding of messages deciding if and how to map messages between different security enclaves.

### 5.4.2 RabbitMQ-AMQP

RabbitMQ (RMQ) [79] is a TCP-based message broker written in Erlang, designed by Pivotal Software, that stores and forwards binary blobs of data. RMQ can be configured to use one of a selection of messaging protocols such as Advanced Message Queuing Protocol (AMQP), STOMP, MQTT, and HTTP. In particular we evaluated AMQP 0-9-1. It is worth noticing that RMQ provides some improvements on top of these basic protocols such as range rejection of messages, geo-based routing, and front-end management.

AMQP [80] implements the classic pub/sub paradigm where brokers receive messages from publishers and route them to consumers. Content delivery can either be push or fetch-based. AMQP supports multiple subscription models which can be summarized as simply sent, topic-based, and tag-based. Metadata can be added to messages either to provide clients with more information or to allow brokers to perform selective forwarding. Reliability is supported utilizing an overlay acknowledge-based protocol and brokers can be configured so that messages are only removed after being acknowledged. When messages are not delivered, AMQP can be configured to send the undelivered message or a lost report back to the publisher. Clients can also reject messages and either ask the broker to reschedule their delivery or signal they will not process them.

Servers and clients monitor their connection using heartbeat messages, if two are lost the TCP connection between them is terminated. Persistency is client-specific and not necessarily supported by all AMQP client implementations, the one we used in our tests [81] was designed to re-issue all the configurations upon performing a re-connection.

RMQ brokers support three clustering mechanisms called Cluster, Federation, and Shovel. The Cluster mechanism provides a centralized logical broker and it's designed for an environment characterized by high availability and throughput. Federation supports the connection of multiple clusters that are far from each other and implements basic mechanisms to overcome latency and disconnections. Shovel provides a customizable layer and it's designed to provide a more rich interface to manage edge cases. Brokers can discover each other either through static IP configurations or by A/AAAA records set in the DNS. Other discovery mechanisms are available but rely on external plugins so they were not considered in this analysis.

For what concerns security, RabbitMQ offers TLS as an encryption layer, and optionally data can be protected through "Virtual Hosts", isolated environments in which AMQP entities are created and kept separated. Using "virtual host", RabbitMQ can provide a full AAA environment where users' rights can be restricted. Authentication is supported either through accounts or by using x.509 certificates.

### 5.4.3 MQTT

MQ Telemetry Transport (MQTT) v3.1.1 [82] is an open OASIS and ISO standard, TCP or WEB-Socket based pub/sub messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. In particular, MQTT was designed to minimize network footprint while providing reliability trades off.

Similarly to RMQ, MQTT allows clients to specify wills, i.e. messages that will be relayed to the publisher by the broker when a client disconnects without notifying the server. Upon subscription, a publisher can decide if messages have to persist client disconnections and the maximum level of delivery guarantees they will accept. MQTT supports once (without confirmation), at least once (with confirmation), and exactly-once (4 ways handshake) semantics. It is worth noting that publishers and subscribers can have a different level of maximum QoS they are willing to accept. In both cases, the QoS level specifies the relation with the broker for the specific topic. Reliability is maintained through an overlay protocol that replies with a pub acknowledge and a pub received respectively for client's publish and receive operations. MQTT also supports per-message retention and the connection with the broker can be set as durable in which case subscription persists client disconnection enforcing delivery of reliable messages.

Client-server connection is monitored using a keep-alive mechanism and clients must periodically send either a ping request or a control message to the server. Losing keep-alive messages for a certain time will cause the endpoints to consider the connection stale and disconnect automatically.

For our experiments we used MOSQUITTO [83] implementation of MQTT. MOSQUITTO supports one-hop forwarding where another broker specifies topics and QoS that will then deliver to subscribed clients.

MQTT provides bare-bone security in the form of encryption.

### 5.4.4 Kafka

Apache Kafka is a TCP-based protocol written in SCALA and JAVA developed by the Apache Software Foundation [84], to provide a unified, high-throughput, and as a low-latency platform to handle real-time data feeds [85]. Kafka exchanges data streams (instead of messages), with topics abstracted as immutable ordered records of data. Kafka stores each client's delivery state as a record position.

Kafka clients connect to a federation of brokers and subscribe or publish to/into channels identified by topics. Consumers and publishers can separately specify the maximum level of QoS that they are willing to support between them and the broker. By default, the information pushed by a publisher is persistent for a configurable retention period that is independent of whether the data was consumed or not by interested peers. Kafka manages reliability using indexes that pair consumers and data. Subscribers send fetch requests to the broker specifying the index of the next piece of information they want. This mechanism can also be used to request older and implement reliability in case of loss. This approach is possible because Kafka divides topics into totally ordered partitions, each of which is consumed only by a specific consumer. Ordering is only guaranteed inside a partition that is specific for a source-topic-consumer. Kafka implements at least-once-delivery by default. Exactly once can be specified either for single consumers or for all of them. Kafka can require all subscribers to acknowledge messages to the publisher.

Kafka implements Synchronization through replication by configuring brokers in a master/slave relation. Publishers send messages to a primary broker that then replicates its status in the replication cluster. Publisher can either require the replication master or, at greater cost, all the replication members to acknowledge their messages. The connection between clients and brokers is defined in KAFKA as the ability of peers to keep connected to their internal database and stay in sync with the partition leader. Being in sync is defined as nodes

maintaining a sufficiently small offset between their topics and the ones of the leader. Kafka discovery is based on gossiping or static configuration. Data is not replicated between different brokers that do not belong to a replication partition.

Kafka natively supports encryption and authorization and manages more complex security scenarios through external plugins and software companions.

### 5.4.5 Redis

Remote Dictionary Server (ReDiS) is an open-source, TCP-based, in-memory data structure store written by Salvatore Sanfilippo and sponsored by RedisLabs [86] which can be used as a database, as a cache, and as a message broker through its pub/sub API.

ReDiS pub/sub protocol is fairly simple. Clients connect to a broker and specify the topic they wish to subscribe to or publish. Messages are stored in a hash map. Reliability follows a fire-and-forget pattern, but if the connection between clients and a server is interrupted (not disconnected), clients and servers store undelivered messages until they can communicate again. ReDiS messages are limited to a maximum size of 512 MB. ReDiS's API implement two methods for receiving messages, the concurrent method is stated to be faster but without ordering guarantees [87].

ReDiS monitors the connection between clients and servers through a simple keep-alive message sent only if more than 300 seconds pass without traffic exchange between two endpoints. If that message is not acknowledged, the connection is considered stale and closed.

Forwarding between clusters is implemented as simple broadcasting between brokers [87].

ReDiS assumes that the entities that can be connected to a broker are trusted and as such it manages encryption through TLS and access control through a configuration-based mechanism. The developers suggest that an ACLs and user input validation layer should protect access to a ReDiS instances.

### 5.4.6 DDS

Data-Distribution Service for Real-Time Systems (DDS) [88] is a data-centric middleware developed to provide pubs/sub messaging in low-latency, critical, and real-time systems.

DDS implements the concept of Data-Centric publish-subscribe where nodes that want to add data register as publishers and nodes that want to access the data as subscribers. DDS is data-centric in the sense that publishers and subscribers must bind the type of data they wish to transmit. This approach separates DDS from other messaging systems that exchange blobs of data. Most of the following information regarding the general working of DDS are extracted from [89]

Applications connecting to DDS interact with a distributed object called Global Data Space that transparently handles the updating and retrieving of information using a publish/subscribe-semantic. Publishers/subscribers univocally bind a data writer/reader (which consequentially also binds the data definition), a topic, and a set of QoS specifications to a specific flow of information with the objective of updating/retrieving data. It is worth noting that a topic identifies a single data type. An application can also attach a listener to each flow

to be notified about data reception, QoS violations, and in general of any asynchronous event relevant to the flow. The connection between publishers and subscribers is modeled as a domain plain where only entities that belong to the same plane can interact with each other.

DDS supports an extensive number of QoS characterization other than reliability and order. For example, it is possible to specify bandwidths and latency deadlines for delivery, control if data must be volatile or persist publishing. Applications have access to an extensive set of configurations to control what happens when data is updated, for example, it is possible to specify that the subscriber is only interested in receiving the latest update or just some updates per unit of time. The interested reader is invited to study the specification to gain a full understanding of the detail of the QoS capabilities [89].

DDS bases its wire protocol on RTPS, a reliable transport protocol for pub/sub communication over unreliable protocols such as TCP, UDP multicast, and unicast [90]. While the specification specifies requirements, interfaces, and data types translation, each vendor-specific implementation has a sufficient degree of freedom so that different implementations are likely not to be compatible unless specific bridges are designed and used.

Clustering and forwarding are implementation-specific, DDS provides a general architecture but it is assumed that each implementation will require a custom approach.

The DDS standard requires the implementation of mechanisms to ensure data integrity, authentication of subscribers and publisher, authentication of messages and data origin, and optionally mechanism to prevent data repudiation. In particular, they propose an extension to the Real-Time Publish-Subscribe (RTPS) Wire Protocol (RTPS) transport protocol [91].

In our experiments, we tested two implementations of DDS: OpenDDS [92], which is a C++ open-source implementation made by the Object Management Group, and RTI-DDS [93], which is a commercial implementation made by RTI.

OpenDDS is built on top of Adaptive Communication Environment (ACE), a platform-independent communication library that OpenDDS exploits for connectivity but also to define the messages that will be supported at run-time through a component called IDL compiler. Applications using OpenDDS must discover each other using one of two discovery options, Information Repository, and RTPS Discovery, or by statically configuring neighbors' addresses. The Information Repository is a centralized infrastructure that runs in a separate process. Publisher and subscribers connect to it and use it to learn about each other. RTPS discovery is distributed on the other hand and based on having each instance advertise its presence to a predefined list of unicast or multicast addresses. Communication between publishers and subscribers can be achieved with TCP, UDP, Multicast IP, and RTPS UDP.

### 5.4.7 DisService

Dissemination Service (DS) [94] is a peer-to-peer communication middleware written in C++, based on multicast, and specifically designed for mobile tactical networks.

DS distributes messages efficiently using a combination of pull and push approaches. Large messages are divided into fragments and proactively shared to increase information availability and liveness. Peers aggres-

sively cache messages even when they are not the recipients, thus becoming able to forward messages that lack a connection between source and destination either in space (not directly connected) or time (connected at different times). By default, DS holds messages as long as storage is available, otherwise switching to a last-recently-used policy to expire messages in the cache.

Client applications can make information available to other nodes in the network by sending messages in a group context. Any other application interested in that information needs to subscribe to that group and select how it wants messages to be retrieved. In particular, a client can specify if it wants messages delivered in publication order or as soon as they are received. Furthermore, clients can decide if they want messages to be reliable or not and can associate priorities to decide from which subscription to receive/add data in case of bandwidth contention. DS's aggressive caching mechanism enables it to store data along the path, shortening the distance that re-transmissions have to traverse in case of corruption or loss.

DS implements reliable multicast communications through a NACK-based mechanism. Subscribing nodes periodically notify their peers of any missing message (or parts of them, termed "fragments" in DS parlance) until whole messages are retrieved.

Receivers can detect missing data either because they only receive fragments of bigger packets or by observing missing sequence numbers from received packets. DS also implements specific mechanisms to share large objects, such as metadata only dissemination and chunking. The first mechanism distributes, by default, only metadata describing the data, only exchanging the data upon request. Chunking instead splits large objects of some known types into smaller ones. For example, one of the chunking algorithms allows DS to send low-quality chunks of an image that can be aggregated to selectively improve the resolution of certain sections. DS also allows for distributed queries to retrieve any data stored in the network.

DS monitors the connection with other nodes through a simple heartbeat mechanism. Upon discovering each other, nodes periodically exchange a heartbeat message to verify the health of the connection between them. Moreover, DS is also capable of configurable levels of message persistence including reliable/unreliable and sequenced/unsequenced. Being based on multicast, DS uses a probabilistic approach to decide when to forward messages to avoid multiple requests to create excessive bandwidth usage but also supports flooding and other heuristics to cover corner cases. DS implements a fully distributed peer-to-peer system and each node can act as a client or broker. DS provides only basic encryption by design, leaving to the upper level the possibility of implementing a more sophisticated group-based security mechanism such as the one we showed in [95].

### 5.4.8 TamTam

TamTam is a communication broker based on top of the UDP Smudge network library [96] which manages group communication and management (discovery, failure detection, and status dissemination). Smudge was designed to be used in constrained environments and allows for the dissemination of small pieces of information (up to the size of one MTU).

TamTam broadcast messages between participants, with nodes forwarding them with at most one level of indirection.

Connections are monitored using status messages characterized by a fixed overhead per group member. These

status messages are piggy-backed on top of ping and acknowledgments. Smudge is designed to provide weak synchronicity between nodes about the status of other nodes, meaning that not all nodes will report the same status about other nodes at a given point in time. Status detection is also not something that is directly delivered to other nodes. Each instance randomly monitors the connection with its neighbors and adds its result to acknowledgment and ping messages used by other nodes to expand their network knowledge.

Being a prototype, TamTam does not provide security mechanisms.

### 5.4.9 ZeroMQ-NORM

ZeroMQ [97] is a network library that can be used to do asynchronous I/O socket programming across multiple transport protocols such as Inter-Process Communication, TCP, and UDP, enabling them to perform fan-out, publish/subscribe, and request-reply communication patterns.

In particular, we configured ZeroMQ to use the NACK-Oriented Reliable Multicast (NORM) [98] transport protocol. NORM, is a UDP-based protocol that supports asymmetrical flows based on unicast or multicast. Contrary to TCP, NORM's reliability is based on negative acknowledgments that are sent by receivers when they don't receive a message. NORM reduces the need of sending negative acknowledgments through the use of Forward Error Correction. NORM also implements a TCP-friendly congestion control algorithm called TCP-Friendly Multicast Congestion Control [99].

Clustering, forwarding, connection monitoring, and security, are left to be built on top of the ZeroMQ abstraction with only some basic interfaces and construct built into the library.

### 5.4.10 JGroups

JGroups [100] is a peer-to-peer group communication toolkit written in JAVA that can either use TCP, UDP unicast, or multicast. Alternatively, JGroups can be extended to use custom transport protocols.

Applications using JGroups specify a group they will use to share information with other nodes. Since JGroups instances can only be connected to a single group, applications must either implement logic to filter messages or start multiple instances to provide something akin to a topic-based publish-subscribe pattern. Since JGroups keeps track of group participants, applications can send messages to single nodes through unicast messages or to the whole group through multicast or unicast messages.

JGroups was designed as a stack of layers, each adding functionalities on top of basic operations such as creating/leaving a group and sending/receiving data to/from other participants. Ordering, reliability (configurable to wait for at least one acknowledge or one sent by all participants), fragmentation, compression, and security, are functionalities that can be added by stacking other layers.

JGroups is a distributed solution that monitors the health of a channel through an overlay protocol designed to verify the status of the group and the connectivity with its participants. Health monitoring is based on sending heartbeat messages and receiving acknowledgments. In particular, JGroups supports two implementations, one based on unicast and one on multicast. The first implementation has each node sending a message to its neighbor on the right, which upon receiving a message will reply with an acknowledgment. The second uses

multicast to do this.

JGroups instances must be part of a group, and group members can find each other using multicast or provide their addresses during the configuration phase. Group members must be able to reach each other directly. JGroups provides facilities to bridge clusters by specifying bridges to separate enclaves. These nodes will forward messages either through unicast or multicast transport. Cluster bridges can automatically be replaced if they crash, provided that the node that replaces them can reach both clusters.

JGroups provides basic security facilities to prevent unauthorized nodes from joining groups or communicating with group members.

### 5.4.11 Edgware Fabric

Edgware [101] is a lightweight message bus developed for Machine-to-Machine communication to support resource-constrained, dynamic, and unreliable environments through a service oriented architecture. Each Edgware endpoint is made of a registry, managing persistency, and a service that acts as the interface for the application through a local MQTT broker or a web interface. These two components allow applications to connect to a federated network of MQTT-compatible brokers which act as a single virtual Publish/Subscribe platform.

Edgware provides three types of communication patterns: Data Feeds, Notifications, and Request/Response. The firsts are data streams pushed onto the bus and intended to be received by multiple consumers. Each consumer must explicitly subscribe to a data feed to receive it. Notifications are ad hoc messages directly sent to other systems. The sender decides which machine will receive each message. Request/Response is a pattern in which a service listens for request messages and replies with a corresponding response message.

Resource discovery is managed through a multicast-based discovery protocol that univocally identifies all the systems/services connected to the federated platform. Systems are then composed of different services that act as producers or consumers.

At the time of writing, the code was the primary source of information for configuration parameters and system behavior and due to time constraints, it was not possible to perform a deeper analysis.

### 5.4.12 GDEM

Generic Data Exchange Mechanism (GDEM) is a proprietary group communication middleware developed by TNO [102] based on the Joint Dismounted Soldier System Information Exchange Mechanism (JDSS) Information Exchange Mechanism [103] specified in STANAG-4677 [104].

GDEM differentiates from JDSS in several ways. For starters, while JDSS implements an XML-based protocol to describe payloads, GDEM is payload agnostic and can evaluate headers without having to parse and decompress the payload. GDEM uses a proprietary protocol called SUSP for message segmentation and transport [105], allowing larger payloads to be broken into variable-sized segments. Moreover, compression is optional in GDEM

Reliability is provided through a repair window that works by having receivers send synchronization requests

upon detecting missing messages. GDEM also supports mechanisms for full synchronization in case messages were lost outside of the repair window.

Clustering and forwarding were added as a simple layer between GDEM and The Simple UDP Segmentation Protocol (SUSP) capable of forwarding messages to other clusters.

## 5.5 Anglova Scenario

The Anglova scenario [14] was developed by the NATO STO IST-124 RTG on "Heterogeneous Tactical Networks - Improving Connectivity and Network Efficiency." The scenario describes the mobility patterns of a military operation conducted in the fictitious area of Fieldomont in Anglova where an allied coalition advances intending to repel an invading force and rescue civilians. This operation is set in a hilly forestry terrain that spans a rectangular area of 13x9 Kilometers. A total of six companies, four mechanized consisting of 24 vehicles, one Command and Artillery of 22, and one Support and Supply with 39, form a battalion of 157 interconnected nodes through a network of VHF and UHF radios.

The Anglova scenario contains mobility patterns vetted by military experts and was designed to highlight the challenges related to establishing necessary services and fulfilling information requirements in heterogeneous networks characterized by resource-limited devices, unpredictable network link-state, and connectivity coupled with node mobility. The scenario in its entirety is composed of 3 vignettes lasting 4 hours, for the experiments, we primarily considered the Troop Deployment Vignette since it presented the most dynamic communication patterns. The interested reader can download this scenario and the related components from https://anglova.net.

### 5.5.1 Troop Deployment Vignette

In the Troop Deployment Vignette, a subset of the battalion consisting of two mechanized and two infantry companies stage an attack against a hostile force advancing into the operational zone.

The stage of this operation is primarily hilly and covered by forests. We extracted the troop mobility patterns from a NATO exercise characterized by movements over a rectangular area of 13x33 km over large and tight roads. The Vignette starts by having the battalion move in a single column over one of the main roads. After about 10 km, the battalion splits up over two main roads, and after 25 more km, it splits up again on many paths grouped in companies. Towards the end, the battalion splits up into platoons.

## 5.6 Synchronized Cooperative Broadcast

Implementing a complete waveform for an emulation environment can be a lengthy process, but there are decisive advantages in experimenting using realistic communication hardware, topologies, and mobility patterns. While emulation may not use real hardware, it can provide a good compromise between realism and price. An emulated scenario can also be better controlled, allowing for multiple repeatable runs that would be prohibitively expensive utilizing real assets. To make the experiments more realistic, the group decided to use SCB, a relatively new model that is gaining traction in many types of scenarios. SCB provides multi-hop communication, low overhead, and it is particularly indicated for broadcast communication and consequently

to TDR operations. In particular, the group implemented two simplified approaches to SCB to improve the quality of the scenario for the experiments within the Anglova scenario and the EMANE network emulation framework.

The first approach uses Simplified Multicast Forwarding (SMF) configured to perform classic flooding where the SMF relay set contains all nodes. This approach enables modeling of a dynamic SCB schedule; however, it also results in more transmissions than a traditional SCB network. Furthermore, this approach is easy to use and only requires the original path-loss values but does not include the gains of cooperative transmissions.

The second approach uses a precomputed SCB topology that mimics a static SCB network and enables full modeling of SCB's cooperative effects. In this method, the sum of receive power from transmitting nodes is compared to a path-loss threshold, $L_{b,"max"}$ to determine the set of nodes that are reachable directly or through multiple hops from a given source node.

Finally, we estimated packet latency based on the number of hops between the source and the destination node, and we set the SCB network data rate to $\frac{R_L}{ND}$, where $R_L$ is the link data-rate, N is the network size, and D is the CB slot size.

### 5.6.1 Emulating SCB

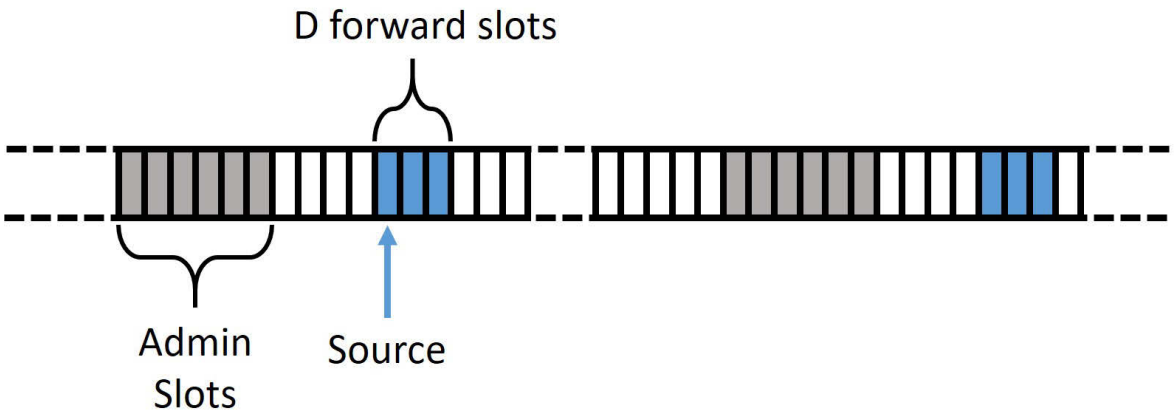



Figure 5.4: SCB slots with dynamic scheduling.

SCB is a waveform primarily designed to transport broadcast traffic within multi-hop mobile networks, making it an interesting choice to support group communication in MANET environments.

SCB works by having all nodes that receive a packet within a time slot retransmit it simultaneously in the next time slot scheduled for the source node's broadcast stream. This retransmission process is repeated until all nodes in the local network have received the packet. The transmission is synchronized in an interval by dividing it into repeated Time Division Multiple Access (TDMA) frames consisting of multiple time slots. The time slots in which nodes transmit or retransmit messages are grouped into an entity called Cooperative Broadcast Slot (CB-Slot), i.e., a group of D time slots. In particular, each slot in a CB-Slot is used to transmit and relay only packets from a single source. Figure 5.4 shows a division in which $D = 3$. As nodes can either receive or transmit packets, the network capacity can never exceed one. Since SCB broadcasts one packet using D time

slots, the network capacity is $\frac{1}{D}$. A bps value can be obtained by multiplying the capacity by the link data rate $R_L$.

The simplified models are based on the idea of approximating the broadcast capacity using parameters set to resemble what would result in a full-scale implementation of SCB. In particular, we considered two approaches, one based on SMF, and one based on precomputing the SCB Topology.

### 5.6.2 SMF Approach

| Number of nodes | Scheduling overhead percentage |
|---|---|
| 24 | 1.1 |
| 48 | 4.2 |
| 72 | 9.5 |
| 98 | 17 |

Table 5.3: Scheduling overhead percentage for networks of various size.

The SMF approach is based on the idea of using SMF and the EMANE's RFPipe radio model. The RFPipe model provides a set of features to emulate waveforms such as data/burst emulation of bandwidth, per-node bandwidth, and sender/receiver per-packet delay control.

SMF is similar to full flooding without simultaneous transmissions. All nodes beside the source relay each packet up to a configurable limit that indicates the maximum number of admissible hops. This approach is based on the Multi Point Relay (MPR) method according to the SMF framework [106]. This method does not implement the TDMA scheduling, and for this reason, it is necessary to estimate the impact of the scheduling overhead $O_s$. The data rate of RFPipe can then be set to $\frac{R_L*(1-O_s)}{D}$, where $R_L$ is the data rate of the link and D is the minimum number of network hops between two nodes simultaneously transmitting different packets. Since this approach mimics a dynamic SCB capable of adapting itself to different traffic loads, this adaptation requires a set of administrative slots as shown in Figure 5.4.

To estimate $O_s$, we assume that the status of each slot can either be owned, blocked, or free. We can then encode this information using two bits. A node owns a slot when it is the only one that can transmit in the CB-Slot. All the other nodes will see the slot as blocked. Finally, a slot is free if no node owns it yet.

The administrative slot must then contain at least 2 * M bits, with M being the length of the schedule that can be adapted. Since each CB-Slot involves D transmissions per frame, the channel resource per node is $\frac{2*D*M}{t}$, where t is the time between administrative slots of the node under examination. Hence for a network with N nodes, with a data rate of $R_L$, a share of $O_s$=$\frac{2*D*M*N}{t*R_L}$ is used for administrative time slots carrying scheduling information.

Table 5.3 summarizes the estimated overhead percentage over the available traffic for different network sizes.

### 5.6.3 Precomputed SCB Topology

Another way to emulate SCB is to precompute the SCB topology and include the cooperative effects when calculating the set of receiving nodes for each transmission. We call this solution Precomputed Synchronized

Cooperative Broadcast (PSCBT), and we implemented it in three steps. First, we built the SCB topology, a map that describes over time whether or not messages transmitted by a source reach a receiver. After that, we characterized the delay of each transmission. Finally, we ensured that the total traffic generated by the nodes never exceeds $\frac{R_L}{D}$.

To build the SCB topology, we compare the power sum obtained by considering all the nodes that are transmitting a message to a path loss threshold $L_{b,max}$ for each other node to decide whether or not the transmission was successful. The SCB topology is the result of performing this calculation for each node as a source. This model relies on the assumption that the SCB transmissions happens in less than one second and that the topology does not change in that time frame.

To calculate the delay, first, we observe that differently from SMF, the schedule does not change, and each node only has one CB-Slot per frame. Consequently, each node has a schedule of M = N * CB-Slot. To model the delay, we assume that the time slots of a CB-Slot are grouped together and that the frame size is M = N * CB-Slot.

Each packet must then wait on a CB-Slot for half a frame for up to four hops, one and a half frames for five to height, and so on. To be accurate, it is also necessary to include the transmission time in the delay estimation, which could be up to four time-slots in the case of four hops. The delay caused by traffic queues in the ingress nodes needs to be measured during the emulation and added to obtain the total transmission delay.

### 5.6.4 Comparison between SMF and PSCBT

The SMF approach has the advantage of modeling a dynamic schedule that can handle traffic changes among the nodes efficiently. This approach only uses the path-loss value between the nodes included in the Anglova Scenario, greatly simplifying the process of modeling networks of different sizes and parameter settings. However, this model lacks the gains of cooperative transmissions, and the delay estimation is not very precise. Furthermore, since messages are relayed through UDP, there is no easy way to support any non-UDP protocol.

On the other hand, the PSCBT approach models a static SCB that has to be pre-configured to the expected traffic loads. Its chief advantage is that it includes the cooperative effects, and the delay is fairly accurate since it only occurs in the ingress nodes. This approach also does not pose limitations to the choice of a transport protocol. The primary disadvantage of PSCBT is that the topology has to be generated for each network size and parameter setting.

## 5.7 Introduction to the experiments

To evaluate the suitability and performance of GCS in TDR operations, we created a test framework using the Anglova scenario, EMANE, and a test harness designed to drive message generation and statistic collection.

To evaluate performance we primarily focused on three metrics:

1. **Message Delivery Latency**, measuring the delay in milliseconds between send and delivery of a message;

2. **Message Delivery Ratio**, expressing the ability of messages to reach recipients;
3. **Bandwidth Consumption**, measuring bandwidth utilization during the experiment.

It is worth noting that since the test harness measures traffic at the senders' sides, reports for certain protocols may be higher than the maximum network throughput. The excessive traffic is not delivered but instead dropped by the network stack. Moreover, the reader should not analyze the three measures in isolation because that could result in misleading conclusions. For example, measuring a low average latency indicates good performance only when paired with a high delivery ratio. This is because the test harness reports delivery latency only for received messages, and those could be few.

The emulation environment was deployed in VMware ESXi using a hybrid EMANE configuration model with the emulated test nodes mapped to a set of networked Virtual Machines (VMs). All emulation components were run by separate EMANE servers' VMs co-located on ESXi host servers.

### 5.7.1 Messaging patterns

Different data types have different dissemination patterns and requirements. Some data types, such as position reports, are generated by each node and are intended to be received by every other node. On the other hand, sensor data is generated by a handful of nodes and may only be needed by a subset of other nodes. Finally, orders and reports are generated by higher echelon nodes (e.g., a Headquarters node) and need to be disseminated down the command hierarchy. In the experiments, we focused on three types of message: Blue Force Data (BFD), Sensor Data (SD), and HQ Documents (Docs). We chose these three as they have different requirements for dissemination and allow the evaluation of different classes of service.

BFD are small messages that identify the current position and status of the transmitting mobile node; they tend to be generated often by each node and need to be received by every other node (at least in the local vicinity or domain). On the other hand, Docs such as Commander's Operational Orders and Intelligence Reports generated by the Operations Center can be much larger (e.g., a document or presentation with multiple slides and embedded graphics) but are not transmitted very frequently, and should only be disseminated to a subset of nodes. In between these two categories lie SD, medium-sized messages typically generated by unattended sensors. They can include high-resolution images or small video segments, which are generated more often than Docs, but not as often as BFD. These messages are generally only consumed by a subset of nodes capable of processing their reports.

### 5.7.2 Test-Harness

To simplify the process of testing multiple protocols, we developed a test harness capable of performing consistent tasks and measuring relative performance. The test harness is written in Java and implements base functionalities for message generation, logging, and configuration, and it consists of a Driver and a StatServer. The Driver exchanges messages with other drivers using the emulated network and reports message transmissions and receiving to the StatServer through a separate high-performance control network.

Each protocol interface was developed as a separate module using PF4J [107], a Java library that enables the implementation of plugins, extension points declared by the application, or loaded at run time. The use of

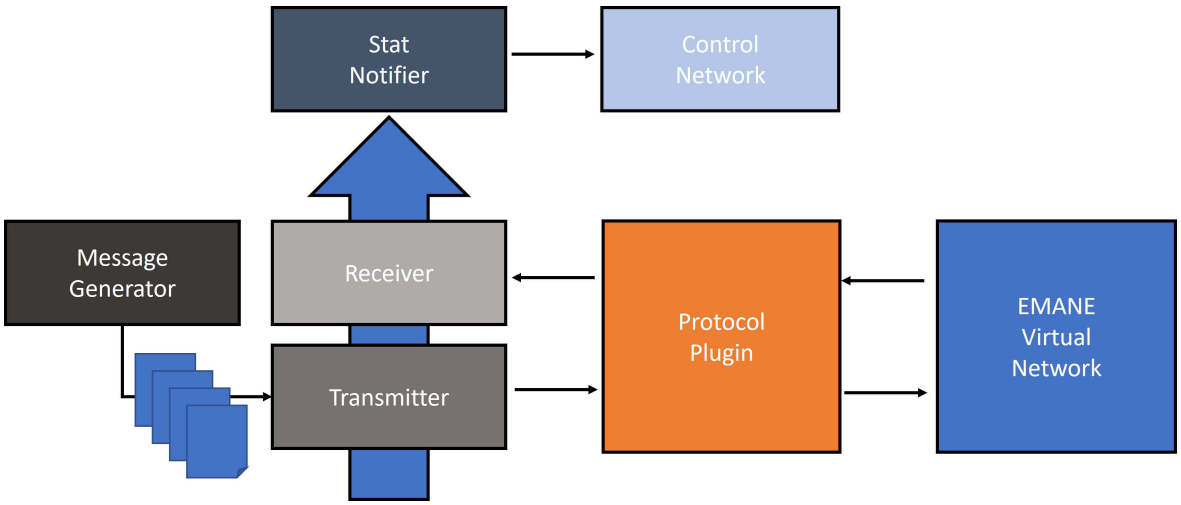



Figure 5.5: Test-Harness Driver Architecture.

this library has several advantages. First, it allows different organizations to develop their plugins in parallel, reducing communication overhead and conflicts. Since the plugins are loaded at run-time, each implementation can remain relatively private, and organizations can deliver obfuscated jars. Another advantage of PF4J is that each plugin can have a separate class loader allowing for a different version of the same library to coexist in a single application. New protocols can be added by creating a new plugin and following a common interface that must override methods to connect, publish, subscribe, and receive messages.

Figure 5.5 shows the Driver architecture made of a Message Generator, a Stat Notifier, a Receiver, a Sender, and a Protocol Plugin. Each time a message is received or sent, the Driver sends a report to the StatServer through a separate high-performance connection. After the scenario has ended, the StatServer outputs a run report.

The connection between the Drivers and the StatServer uses NATS over a dedicated high-performance Ethernet LAN independent from the network managed by EMANE that allowed us to obtain very low jitter and latency for log collection. In particular, this also allowed us to forego time synchronization issues between nodes by having the StatServer log report reception times and use that information to evaluate events latency.

The test harness ensures that all the tests are run consistently and that the same number of messages of each type are generated at the same times by the same nodes within the Anglova scenario. This ensures a fair comparison among the examined protocols. Furthermore, event logging is protocol-agnostic, meaning that regardless of the underlying protocol, the test platform tracks actions such as message transmissions and receptions, generating diagnostic messages that are delivered to the StatServer immediately.

Each report contains publisher and subscriber UUIDs, a message Type, and an incremental number. The first two UUIDs are strings that uniquely identify the node that published the message and the one that received it. We used the incremental number to uniquely identify each message based on type and source so that we could pair subscribers' and publishers' reports. We used the reports at the end of the scenario to extract two metrics: delivery ratio and delivery latency.

The delivery metric measures a protocol's ability to deliver information to the recipients. A delivery ratio of 100% implies that every message that was generated reached all its destinations. The latency metric measures

the average delay (in milliseconds) between when a message was generated and received.

We measured each protocol bandwidth utilization by deploying the tcpdump utility [108] in each node.

We collected delivery ratio and latency results with three different time cut-offs – 5 seconds, 10 seconds, and end of the scenario. These cut-offs represent different tolerances for latency. For example, the policy might be that BFD that has a latency over 10 seconds is too old and hence unusable.

For what concern message patterns, unless differently stated, BFD was 128 bytes in size and sent by everyone to everyone every 10 seconds. SD to 128 KB every 180 seconds, sent by node six and received by nodes 1, 2, and 4. Finally, Docs were 512 KB in size, sent by node four, and received by nodes 1 and 2 every six minutes.

## 5.8 Experiments: 802.11ah

In this experiment, we considered the first 20 minutes of the ”Troop Deployment” vignette [109]. In this vignette, a single company of 24 mechanized nodes stages an attack against insurgent forces advancing into the coalition zone. The vignette starts with the nodes moving from the headquarters to the operation area. Each node uses a 1MHz-600 Kbps wideband radio, creating a network where each node can typically reach another node in less than three hops. We emulated the radio model using EMANE by modifying an already implemented 802.11abg into 802.11ah. Since we simulated a low bandwidth network, we disabled routing and relied on the GCSs' capabilities.

We set up brokers on nodes 1 and 2 for protocols that required centralized brokers, such as RabbitMQ, NATS, and Redis. We picked those two nodes because they presented the best connectivity to the others. Nodes 1 and 2 were also the only nodes able to communicate with other companies, making them a meaningful choice for a centralization point in a real operation. The Distributed protocols (OpenDDS, TamTam, ZeroMQ, and DS) had their brokers/service/daemon deployed in each node.

In this experiment, we only disseminated BFD messages. Each message was 128 bytes in size and generated every 10 seconds by each node. Since all nodes were subscribed to each message, the experiment would at least generate 24 * 128 = 3072 bytes every second, a number well below the maximum radio rate of 600 kbps.

For this experiment, we selected seven group communication solutions from the ones described in Section 5.4: NATS, OpenDDS, TamTam, RabbitMQ, ZeroMQ-NORM, Redis, and DS. In particular, for DS we harvested two sets of results, one where BFD were sent reliably and one where they were sent unreliably. The others sent them reliably.

### 5.8.1 Delivery Ratio

Figure 5.6 shows the mean delivery ratio for each GCS, in the first 5 seconds, in the first 10, and overall for the duration of the experiment. From the picture, it is possible to see that DisService with reliability enabled (DS-R) achieves a good delivery ratio followed by ZeroMQ+NORM (ZMQN) and Redis. We hypothesized that OpenDDS's terrible performance could be caused by a missing plugin capable of handling resource-constrained environments. TamTam fails to deliver messages in the beginning but achieves better delivery as time goes on.

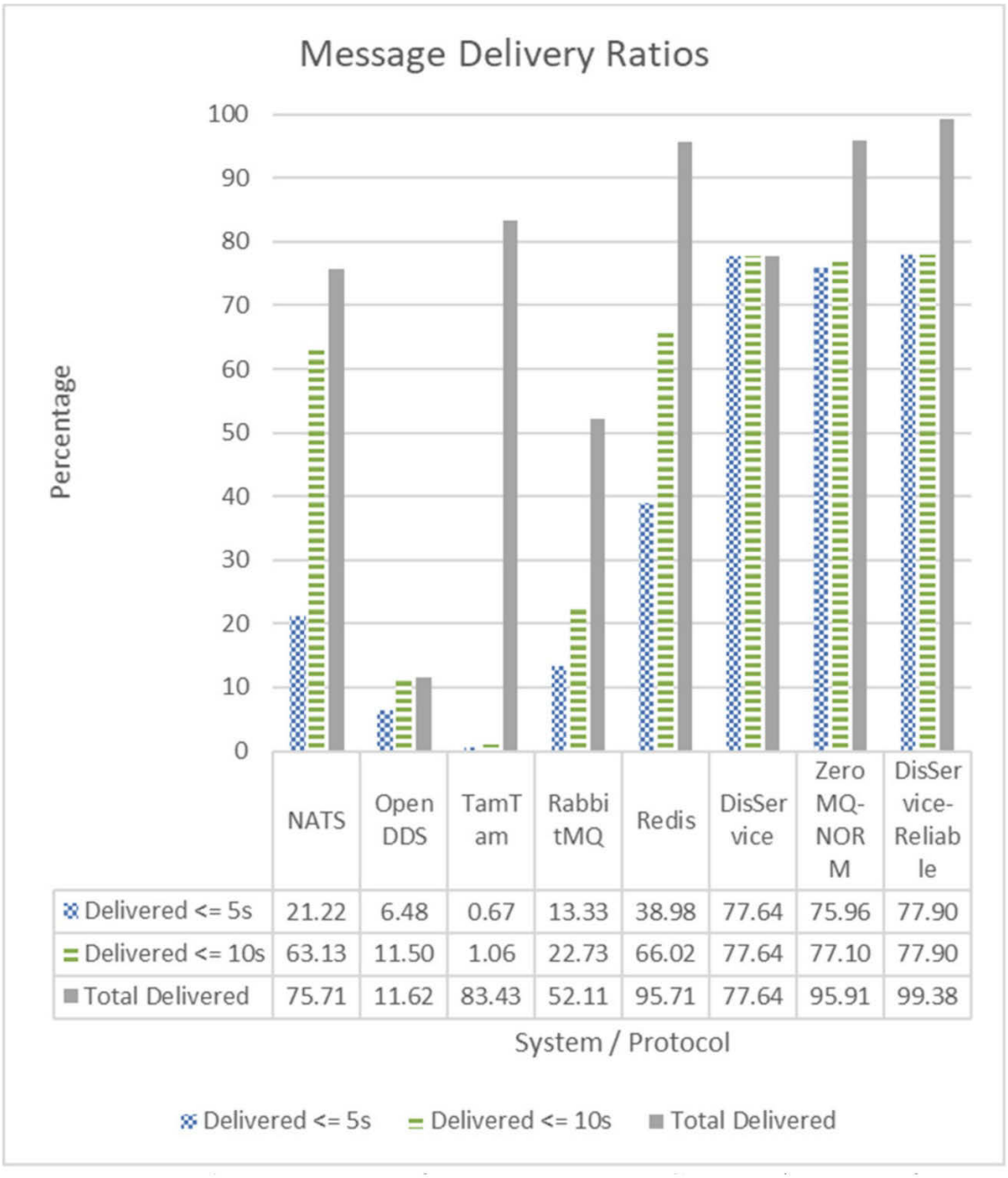



Figure 5.6: Mean Delivery ratio per system/protocol.

Figure 5.7 shows the average number of nodes receiving messages over time. Ideally, each node would receive at each tick all the messages sent by other nodes (23 messages for each tick). Similar to what was shown in the previous picture, DS, Redis, and ZMQN all achieve good performance. From the figure, it is also possible to observe that halfway through the experiment, the GCSs without reliability and the centralized ones have a dip in performance. This is likely caused by a general disconnection that characterized that period. The reason for the drop, in the end, is that since the scenario ends, there is not enough time for the reliability mechanism of each protocol to finish delivering messages.

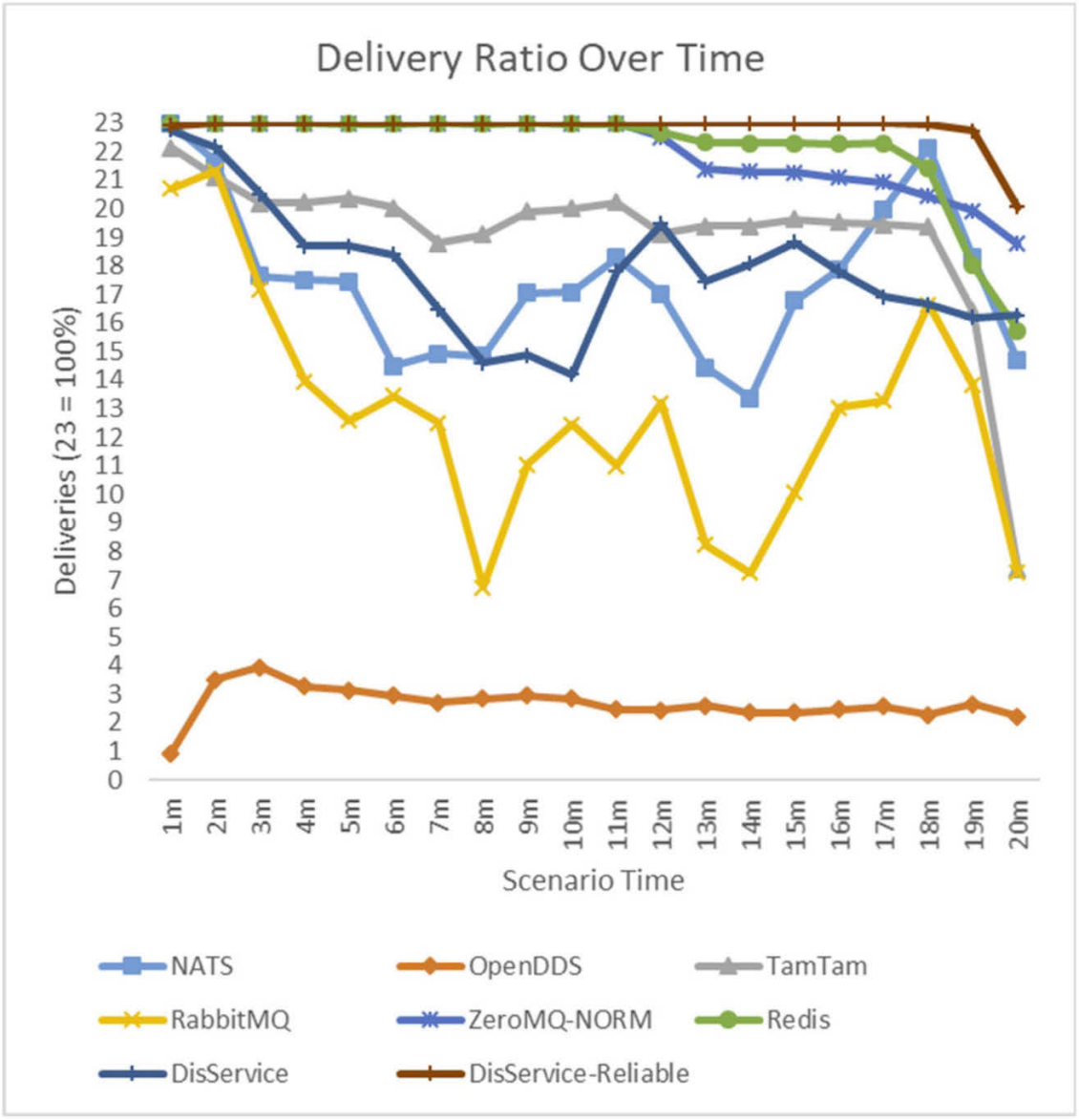



Figure 5.7: Delivery Ratio Over Time per solution/protocol.

### 5.8.2 Delivery Latency

| Protocol | Delivered $\leq$ 5s | Delivered $\leq$ 10s | Total Delivered |
|---|---|---|---|
| NATS | 1831.85 | 5206.08 | 13998.33 |
| OpenDDS | 4173.84 | 5005.14 | 5067.83 |
| TamTam | 2558.98 | 4364.24 | 70181.49 |
| RabbitMQ | 2218.56 | 4424.88 | 15750.49 |
| Redis | 1883.24 | 3858.83 | 50874.49 |
| DS | 97.23 | 97.23 | 97.23 |
| ZeroMQ-NORM | 105.25 | 222.48 | 27265.64 |
| DS-Reliable | 94.99 | 94.99 | 8754.96 |

Table 5.4: Average latency in milliseconds for each message delivery class.

Figure 5.8 and table 5.4 show the average latency of message delivery for each GCS. Note that while the vertical

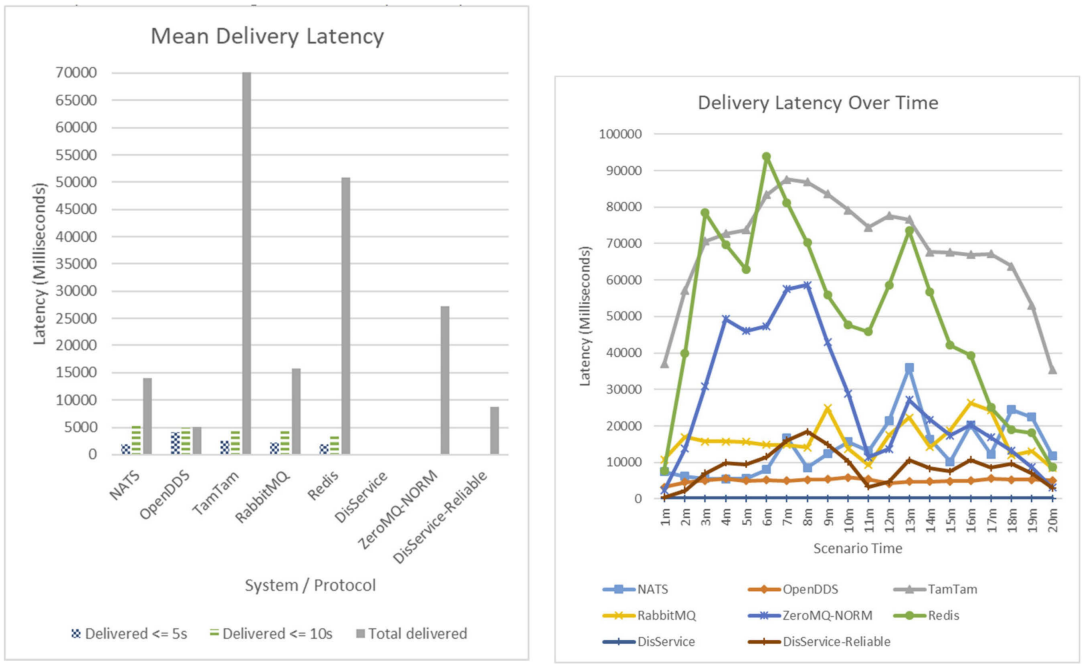



Figure 5.8: Average latency of message delivery (left), and latency over time (right).

axis shows the average latency for delivered messages, lower latency does not necessarily mean better performance because undelivered messages, were not included in this count (as an example, consider DisService without reliability enabled (DS-R), the latency is the same for all three experiments because either a message was delivered or was lost, similarly, OpenDDS has very low latency, but its delivery rate was low).

Figure 5.8 also shows the average delivery latency over time. From this figure, it's possible to see that certain protocols have significant spikes during the experiment. Since they don't have reliability mechanisms, unreliable protocols present more stable latencies.

### 5.8.3 Bandwidth

Figure 5.9 shows the average bandwidth utilization of each protocol in Kbps. This metric is particularly important because TDR environments are generally characterized by tight bandwidth constraints. Bandwidth was measured by capturing outgoing traffic in every node through the tcpdump utility. It is worth noting that not all the outgoing traffic would reach its destination due to network unreliability.

From the figure, we can see that DS-R is the most bandwidth-efficient protocol followed by DS-R and TamTam. It is worth noting that DS-R simply performs UDP multicast with some overhead (without overhead, data exchange should only use 2.4Kbps).

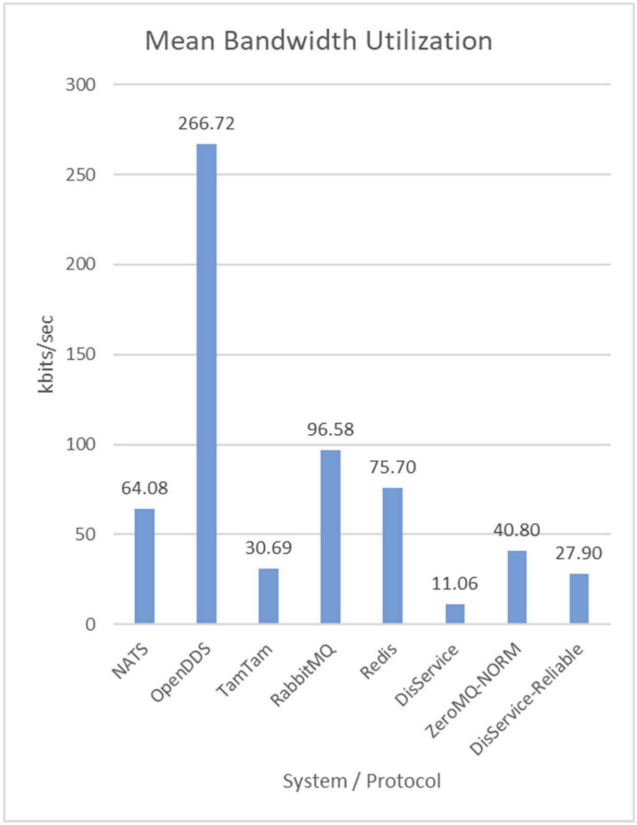



Figure 5.9: Average bandwidth utilization in Kbps.

## 5.9 Experiments: LAN VS 802.11ah

This second set of experiments builds on top of the previous one by adding protocols and by incorporating new message types: SD and Docs. In particular, to the protocols previously included (NATS, RabbitMQ, DS, OpenDDS, Redis, and ZeroMQ), we also added Kafka, MQTT, and Edgware Fabric. During the experimentation, we had to drop TamTam and Edgware fabric. The first could not manage big messages, and the second failed to scale to 24 nodes. Kafka's performance was also bad. The behavior that we observed was that the broker would receive and acknowledge messages but then deliver them with increasing latency until no message was delivered anymore.

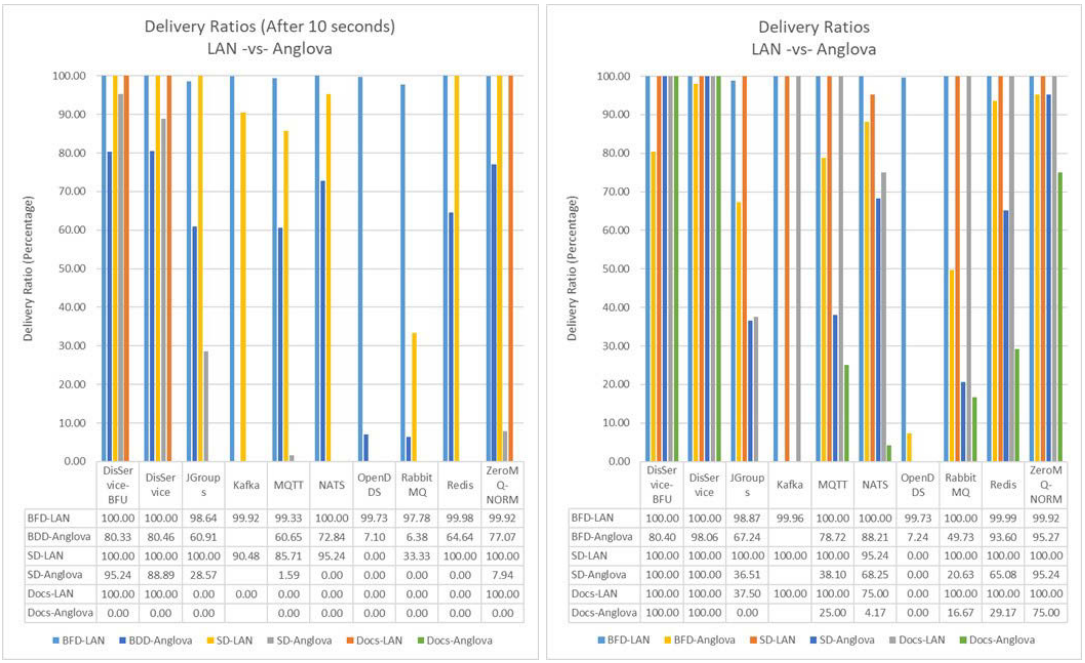
First, we evaluated the GCSs in a traditional wired environment (an Ethernet Local Area Network (LAN)) to provide baseline results and then in the Anglova scenario. Similar to the previous experiment, we compared the protocols by measuring the delivery ratio, latency, bandwidth consumption, and average cost per message.

In this experiment, we had three different types of messages: BFD, SD, and Docs. BFD size and frequency did not change from the previous experiment. SD messages consist of 128 KBytes each, are transmitted by only one node (the sensor gateway) every 3 minutes (for a total of 21 messages during the simulation), and are supposed to be received by only 3 nodes. Finally, Docs consists of 512 KBytes each, are generated only by one node (node 4) every six minutes, and are intended to be received only by two nodes (node 1 and 2).

The latency measurements are separated first, based on the message type, and then on whether they were collected in the Anglova or LAN experiment. Similar to the previous set of experiments, we harvested two sets of results for DS, one with BFD sent reliably and sequenced (denoted with DS), and one unreliably and

sequenced (with DS-BFU). Note that DS was configured to send SDs and Docs reliably and sequenced in all experiments.

### 5.9.1 Delivery Ratio




| | DisService-BFU | DisService | JGroups | Kafka | MQTT | NATS | OpenDDS | RabbitMQ | Redis | ZeroMQ-NORM |
|---|---|---|---|---|---|---|---|---|---|---|
| BFD-LAN | 100.00 | 100.00 | 98.64 | 99.92 | 99.33 | 100.00 | 99.73 | 97.78 | 99.98 | 99.92 |
| BDD-Anglova | 80.33 | 80.46 | 60.91 | | 60.65 | 72.84 | 7.10 | 6.38 | 64.64 | 77.07 |
| SD-LAN | 100.00 | 100.00 | 100.00 | 90.48 | 85.71 | 95.24 | 0.00 | 33.33 | 100.00 | 100.00 |
| SD-Anglova | 95.24 | 88.89 | 28.57 | | 1.59 | 0.00 | 0.00 | 0.00 | 0.00 | 7.94 |
| Docs-LAN | 100.00 | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 100.00 |
| Docs-Anglova | 0.00 | 0.00 | 0.00 | | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

| | DisService-BFU | DisService | JGroups | Kafka | MQTT | NATS | OpenDDS | RabbitMQ | Redis | ZeroMQ-NORM |
|---|---|---|---|---|---|---|---|---|---|---|
| BFD-LAN | 100.00 | 100.00 | 98.87 | 99.96 | 100.00 | 100.00 | 99.73 | 100.00 | 99.99 | 99.92 |
| BFD-Anglova | 80.40 | 98.06 | 67.24 | | 78.72 | 88.21 | 7.24 | 49.73 | 93.60 | 95.27 |
| SD-LAN | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 95.24 | 0.00 | 100.00 | 100.00 | 100.00 |
| SD-Anglova | 100.00 | 100.00 | 36.51 | | 38.10 | 68.25 | 0.00 | 20.63 | 65.08 | 95.24 |
| Docs-LAN | 100.00 | 100.00 | 37.50 | 100.00 | 100.00 | 75.00 | 0.00 | 100.00 | 100.00 | 100.00 |
| Docs-Anglova | 100.00 | 100.00 | 0.00 | | 25.00 | 4.17 | 0.00 | 16.67 | 29.17 | 75.00 |

Figure 5.10: Delivery ratios in the first 10 seconds (left), and after 20 minutes (right).

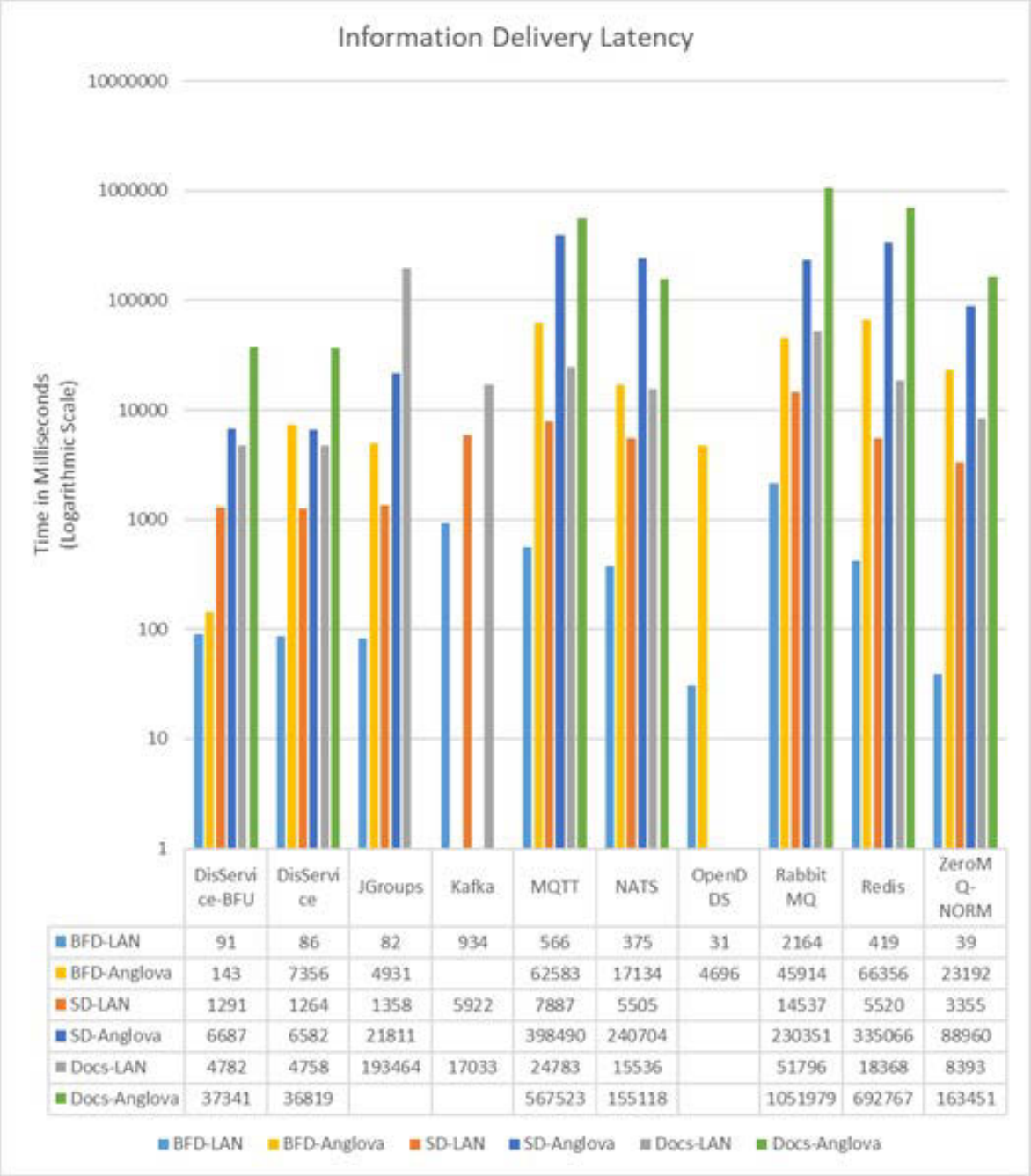
The left plot of Figure 5.10 shows the delivery ratios in the first 10 seconds for each protocol during the LAN and Anglova experiments. This figure shows that most protocols performed well under the LAN environment validating our configuration. BFD messages were correctly delivered by most protocols. OpenDDS and RabbitMQ had significant problems handling SD messages in the LAN simulation. Only two protocols, DS and ZMQN were able to handle Docs in the LAN environment. Note that messages, in this case, are counted as delivered only if their latency is smaller than 10 seconds.

Conversely, performance results obtained during the experimentation in the Anglova scenario were much worse. Kafka performing especially badly, unable to deliver anything. No protocol was able to deliver Docs in the interval considered. DS was the only protocol capable of delivering most SD messages. Finally, DS, ZMQN, NATS, Redis, JGroups, and MQTT delivered most BFD messages.

The second plot of Figure 5.10 shows the delivery ratios at the end of the experiment (which lasted 20 minutes). Having more time, most protocols can deliver more. DS, ZMQN, and Redis, all achieve over 90% of delivery of BFD messages. For SD messages, DS and ZMQN are over 95% while all the other protocols are below that. For Docs, DS and ZMQN have the best performance. DS is again one of the best protocols able to deliver most messages with the smallest latency.

### 5.9.2 Delivery Latency

Figure 5.11 shows the average delivery latency on a logarithmic scale. This choice was made to compensate for the large variance in observed latency. In the LAN environment, OpenDDS has the lowest latency for BFD




| | DisService-BFU | DisService | JGroups | Kafka | MQTT | NATS | OpenDDS | RabbitMQ | Redis | ZeroMQ-NORM |
|---|---|---|---|---|---|---|---|---|---|---|
| BFD-LAN | 91 | 86 | 82 | 934 | 566 | 375 | 31 | 2164 | 419 | 39 |
| BFD-Anglova | 143 | 7356 | 4931 | | 62583 | 17134 | 4696 | 45914 | 66356 | 23192 |
| SD-LAN | 1291 | 1264 | 1358 | 5922 | 7887 | 5505 | | 14537 | 5520 | 3355 |
| SD-Anglova | 6687 | 6582 | 21811 | | 398490 | 240704 | | 230351 | 335066 | 88960 |
| Docs-LAN | 4782 | 4758 | 193464 | 17033 | 24783 | 15536 | | 51796 | 18368 | 8393 |
| Docs-Anglova | 37341 | 36819 | | | 567523 | 155118 | | 1051979 | 692767 | 163451 |

Figure 5.11: Average latency of information delivery.

messages followed by DS. This result is reversed in the Anglova experiment where DS-R performs best and by a much wider margin. DS-R is still a top contender but presents a higher latency than OpenDDS and JGroups. This experiment highlights the trade-off between reliability and latency. DS-R can deliver more than 80 percent of data with very low latency while DS-R can deliver around 99% of messages but with a decisive increase in latency (more than 60x).

For SD messages, DS appears to be the best both in the LAN and Anglova experiments by a huge margin compared to the other protocols. For Docs, DS is the only protocol able to deliver 100% of messages in the Anglova scenario followed by ZMQN with its 75% delivery ratio. In the LAN experiments, DS is still the best performer with 4758 ms of latency followed by ZMQN with 8393ms, and NATS with 15536ms.

### 5.9.3 Bandwidth

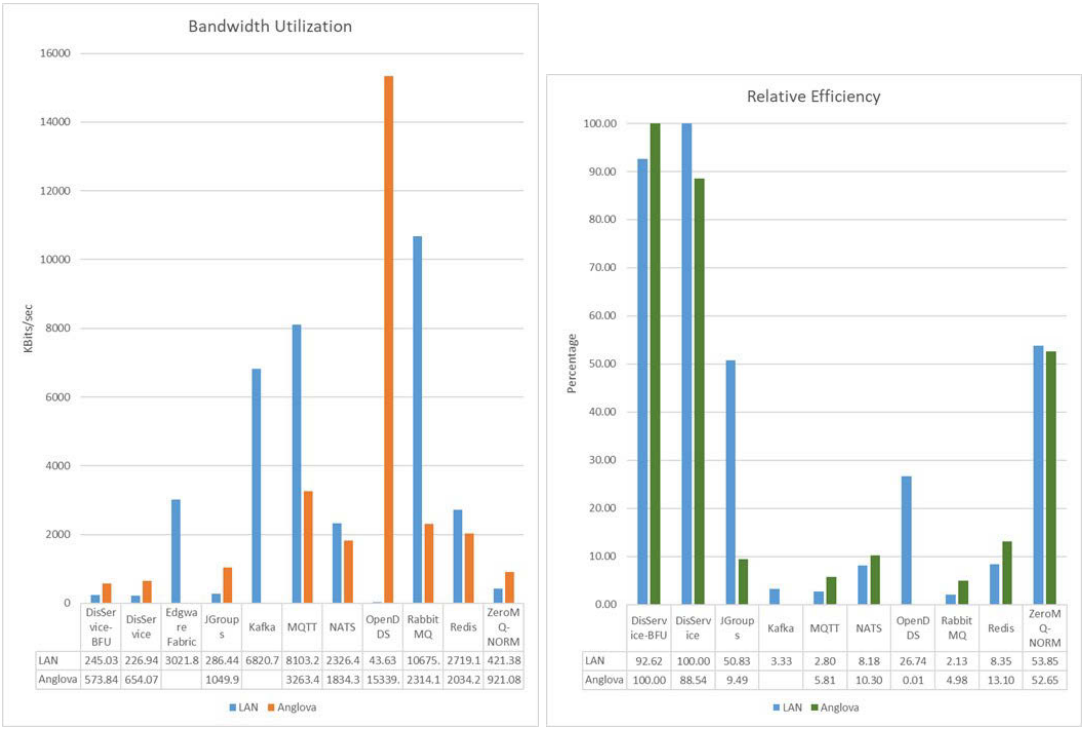



Figure 5.12: Bandwidth utilization (left), and efficiency (right).

The left plot of Figure 5.12 shows the bandwidth utilization in Kbps for both LAN and Anglova environments. While OpenDDS has the lowest bandwidth utilization in the LAN environment, the protocol was also only able to deliver BFD messages. A similar observation is true for JGroups which delivers much fewer messages than the competitors. The next best performer is DS followed by ZMQN. The worst performer in terms of bandwidth consumption appears to be RabbitMQ. In the Anglova scenario, all protocols show a distinct increase in bandwidth consumption due to the increased necessity of retransmitting lost messages. The best performers, in this case, are unreliable and reliable DS followed by ZMQN.

The right plot of Figure 5.12 shows the relative bandwidth efficiency of each protocol obtained by calculating the ratio between delivered messages and bandwidth cost normalized to the performance of the best performing protocol. We set the efficiency of the best protocol to 100% and compared the others based on how close they matched its performance. It is worth noting that this measure compares delivery over bandwidth and not over latency, and as such, it is not an all-encompassing measure of protocol performance since depending on the application latency or delivery ratio can be more significant.

In the LAN environment, the best performer is DS-R followed by DS-R, ZMQN, and JGroups. The worst protocols in this analysis were RabbitMQ, MQTT, and Kafka. In the Anglova environment, the best performer was DS-R followed by DS-R and ZMQN. It can be seen that reliability, in this case, was much more expensive than in the LAN environment. In this experiment, OpenDDS was the worst performer likely because this DDS implementation is not optimized for degraded environments.

## 5.10 Experiments: SCB - SMF vs PSCBT

In the third set of experiments, we replaced 802.11ah with two SCB models using the same configurations used in [109], with $L_{b,max}$ set to 139 dB, a 1.25 MHz bandwidth, and with a link data rate of 875 kbps. Figure 5.13 shows movement patterns of each company in the scenario.

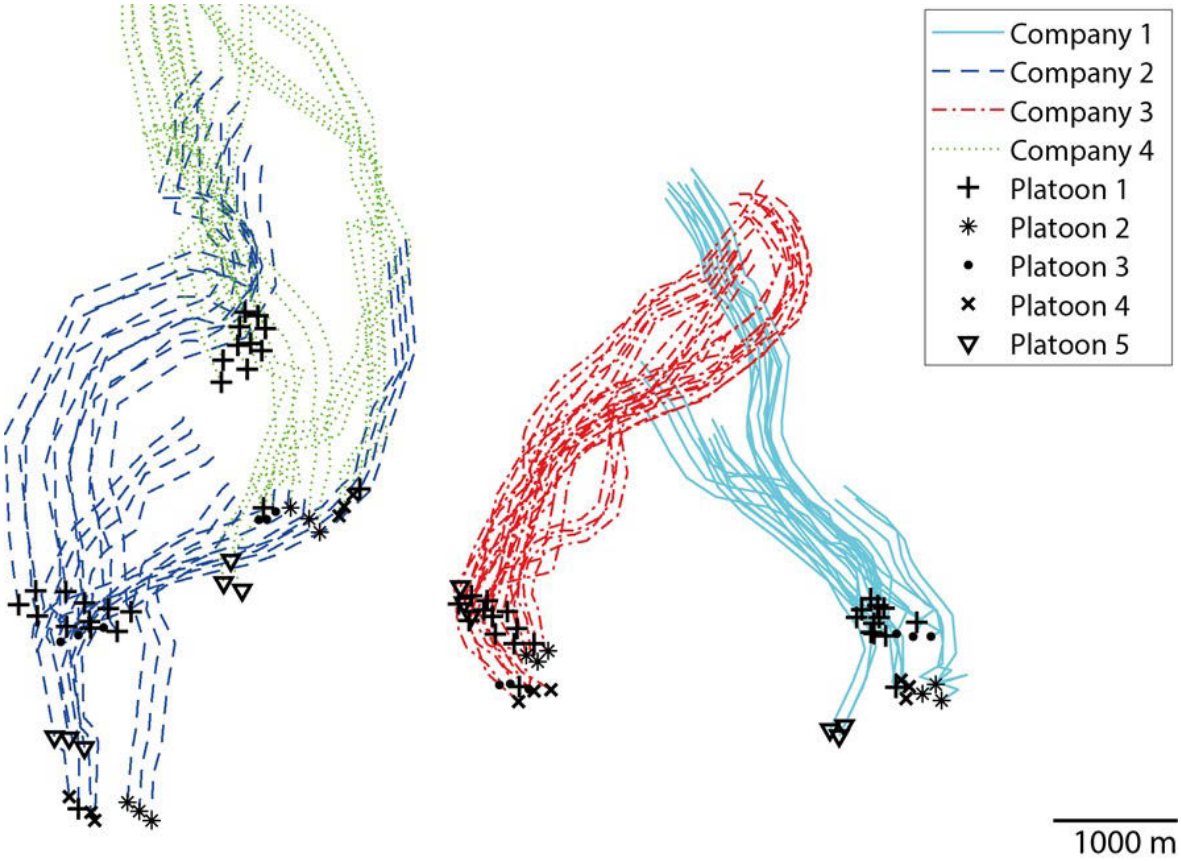



Figure 5.13: Movement of each company in the Anglova scenario.

| Traffic Pattern | Size | Frequency |
|---|---|---|
| **Low** | 128 Bytes | 2 Seconds |
| **Medium** | 1024 Bytes | 2 Seconds |
| **High** | 1024 Bytes | 1 Seconds |

(a) 24 Nodes

| Traffic Pattern | Size | Frequency |
|---|---|---|
| **Low** | 128 Bytes | 2 Seconds |
| **Medium** | 1024 Bytes | 4 Seconds |
| **High** | 1024 Bytes | 2 Seconds |

(b) 48 Nodes

Table 5.5: Summary of SMF data rates for 24 and 48 nodes.

| Traffic Pattern | Size | Frequency |
|---|---|---|
| **Low** | 128 Bytes | 2 Seconds |
| **Medium** | 1024 Bytes | 6 Seconds |
| **High** | 1024 Bytes | 3 Seconds |

(a) 72 Nodes

| Traffic Pattern | Size | Frequency |
|---|---|---|
| **Low** | 128 Bytes | 2 Seconds |
| **Medium** | 1024 Bytes | 8 Seconds |
| **High** | 1024 Bytes | 5 Seconds |

(b) 96 Nodes

Table 5.6: Summary of SMF data rates for 72 and 96 nodes.

In the third set of experiments, we set for each configuration the number of CB-Slot that can be dynamically allocated, to M = 2 * N, that t is one second, and that D is 4.

| Traffic Pattern | Size | Frequency |
|---|---|---|
| **Low** | 128 Bytes | 5-10 Seconds |
| **Medium** | 512 Bytes | 5-10 Seconds |
| **High** | 1024 Bytes | 5-10 Seconds |

Table 5.7: Summary of PSCBT data rates.

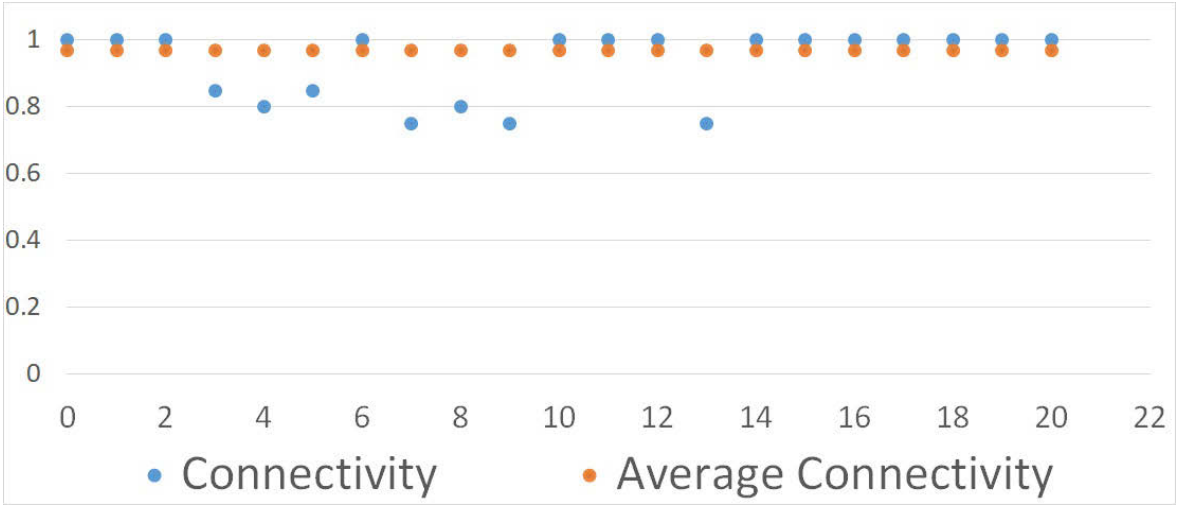



Figure 5.14: Interpolated theoretical precomputed connectivity for the 24 nodes network

This set of experiments is made of several scalability tests with networks of 24, 48, 72, and 96 nodes. In particular, the 24 nodes experiment consists of just Company 1. The 48 nodes experiment consists of Company 1 and 3, the 72 nodes of Company 1, 3, and 4, and the 96 nodes experiment of all four companies. For the experiments with 24 nodes, we used the first 20 minutes of the Anglova Scenario, while for the three larger configurations we added from second 5500 to 6501.

Table 5.5, 5.6, and 5.7 show the different messaging patterns that we tested. Note that while to simplify the experiment we only transmitted BFD, we actually varied traffic load and transmission intervals to test different communication patterns.

In this configuration, we conducted three sets of experiments. The first set was done to evaluate the performance of the SMF approach with 24, 48, 72, and 96 nodes with low, medium, and high levels of traffic. The second set is to compare the SMF and the PSCBT approaches with a 24 node configuration. Finally, in the third set of experiments, we evaluated the performance of four different GCS plus a baseline IP Multicast over UDP implementation using the PSCBT model.

Figure 5.14 summarizes the theoretical network connectivity obtained from the PSCBT which is defined as the fraction of node pairs that are connected via one or more hops. The value shown (0.97 for the 24 nodes network and 1 for the larger networks) should be similar to the delivery ratios obtained during the emulation.

### 5.10.1 Delivery Ratio

Figure 5.15 to 5.18, show the delivery ratio throughout the scenario. The first observation is that there are some differences in both SCB models between the emulation delivery ratio and what was shown in Figure 5.14.

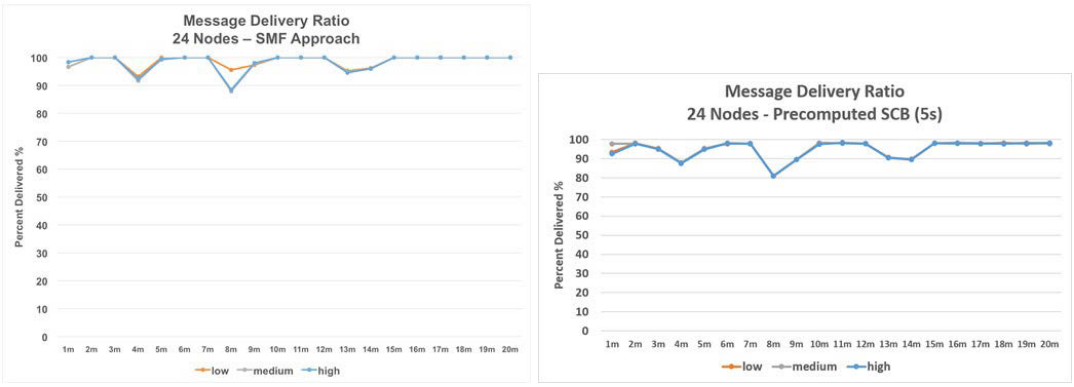



Figure 5.15: SMF (left) vs PSCBT (right) - Message Delivery Ratio for 24 Nodes.

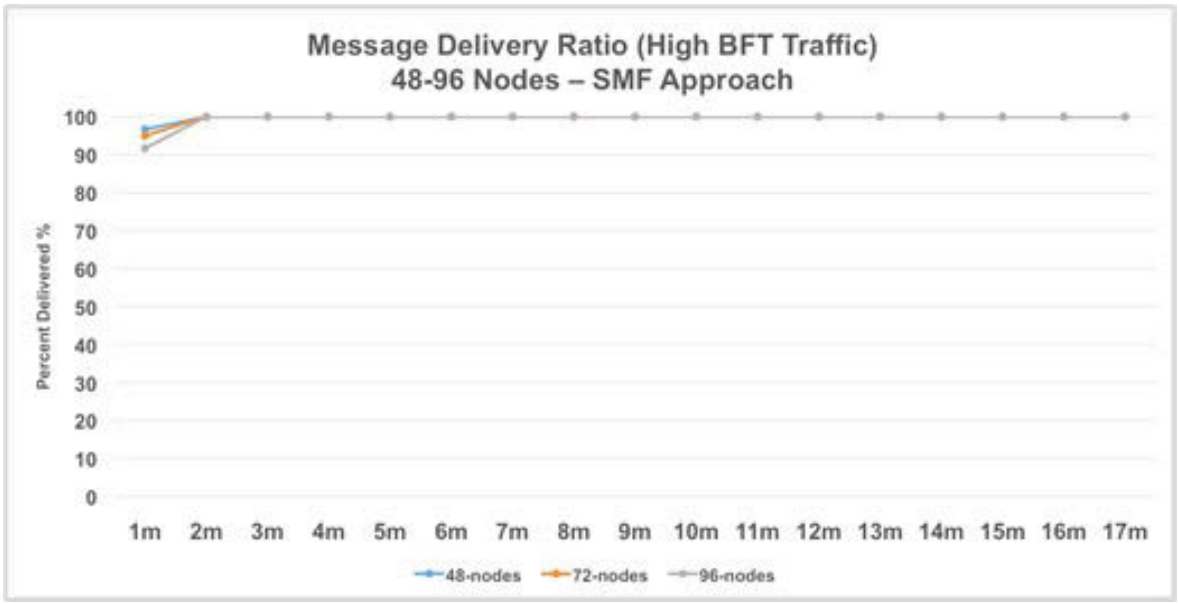


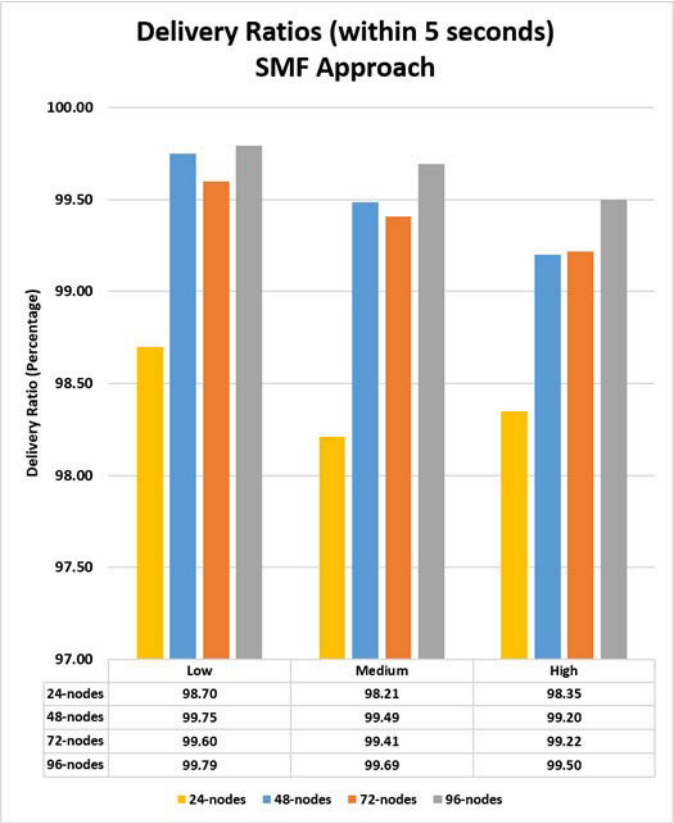

Figure 5.16: SMF Approach Message Delivery Ratio - 48 to 96 Nodes.

In the 24 nodes network, at minutes 4, 8, and 14, there are several performance drops in the SMF delivery ratio for all traffic loads, as shown in Figure 5.15. Figure 5.14 shows that these drops correspond to known sections of the scenario where node fragmentation occurs.

For larger networks, we would expect a delivery ratio of 100% since all the nodes should be fully connected throughout the scenario thanks to rebroadcasting. Hence, the slightly lower ratio that can be seen in the first two minutes in Figure 5.16 will need to be investigated.

Figure 5.17 shows the delivery ratio for the SMF approach on different networks and traffic loads. From the picture, it is possible to see that the delivery ratio is slightly higher with 24 nodes than the 0.97 theoretical bound for SCB. This may be an effect of SMF flooding that causes nodes to retransmit packets at a larger time diversity than the one that could be possible in an actual SCB implementation.

Figure 5.18 show the delivery ratio and GCS results for the PSCBT approach. From the figure's subplots, it is possible to see that DS-R and the baseline SCB Multicast implementation (basically IP Multicast over UDP) have similar performance and are close to the theoretical computed delivery ratio of 97%. RTI-DDS provides similar results but only when sending messages of 128 bytes. This limitation will need to be investigated in




| | Low | Medium | High |
|---|---|---|---|
| 24-nodes | 98.70 | 98.21 | 98.35 |
| 48-nodes | 99.75 | 99.49 | 99.20 |
| 72-nodes | 99.60 | 99.41 | 99.22 |
| 96-nodes | 99.79 | 99.69 | 99.50 |

Figure 5.17: SMF: Delivery Ratio for Messages Received Within 5 Seconds.


| | DisService | DisService Unreliable | NATS | RTIDDS | ZeroMQ-NORM | SCB-Multicast |
|---|---|---|---|---|---|---|
| BF-128-5sec | 94.31 | 94.91 | 22.87 | 94.61 | 90.15 | 94.91 |
| BF-512-5sec | 92.27 | 95.15 | 9.23 | | 88.26 | 95.12 |
| BF-1024-5sec | 91.32 | 95.15 | 7.24 | | 82.76 | 94.80 |
| BF-128-10sec | 95.02 | 95.18 | 82.46 | 95.29 | 94.14 | 95.22 |
| BF-512-10sec | 93.90 | 95.18 | 7.34 | | 93.13 | 95.28 |
| BF-1024-10sec | 94.42 | 95.19 | 8.10 | | 92.15 | 95.09 |

| | DisService | DisService Unreliable | NATS | RTIDDS | ZeroMQ-NORM | SCB-Multicast |
|---|---|---|---|---|---|---|
| BF-128-5sec | 99.98 | 94.91 | 77.65 | 100.00 | 100.00 | 94.91 |
| BF-512-5sec | 99.98 | 95.15 | 47.32 | | 100.00 | 95.12 |
| BF-1024-5sec | 99.98 | 95.15 | 26.97 | | 100.00 | 94.80 |
| BF-128-10sec | 99.97 | 95.18 | 100.00 | 100.00 | 100.00 | 95.22 |
| BF-512-10sec | 99.97 | 95.18 | 36.46 | | 100.00 | 95.28 |
| BF-1024-10sec | 99.97 | 95.19 | 44.40 | | 100.00 | 95.09 |

Figure 5.18: PSCBT: Delivery ratios in the first five seconds (left) and by the end of the scenario (right).

future work. DS-R and ZMQN obtain delivery ratio close to 100% at the end of the scenario. Note that both protocols provide reliable multicast.

NATS is the only solution we considered for the experiments which relyed on TCP and a centralized broker. From the figures, it is possible to see that the only configuration in which it provides a reasonable delivery ratio (82%) is with the smallest BFD of 128 bytes each and at the lowest update rate (10 seconds). This is likely caused by the high cost of unicast traffic.

### 5.10.2 Delivery Latency

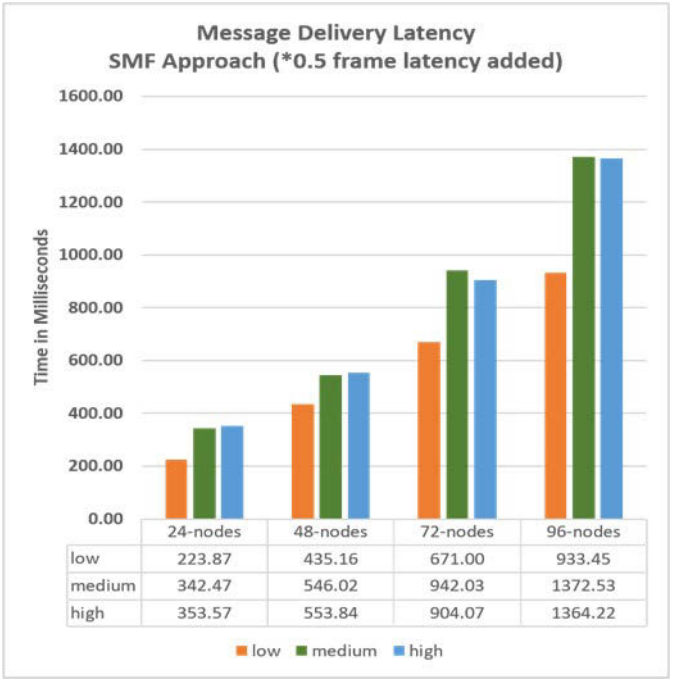



Figure 5.19: SMF Approach - Message Latency With 0.5 Frame Latency Added.

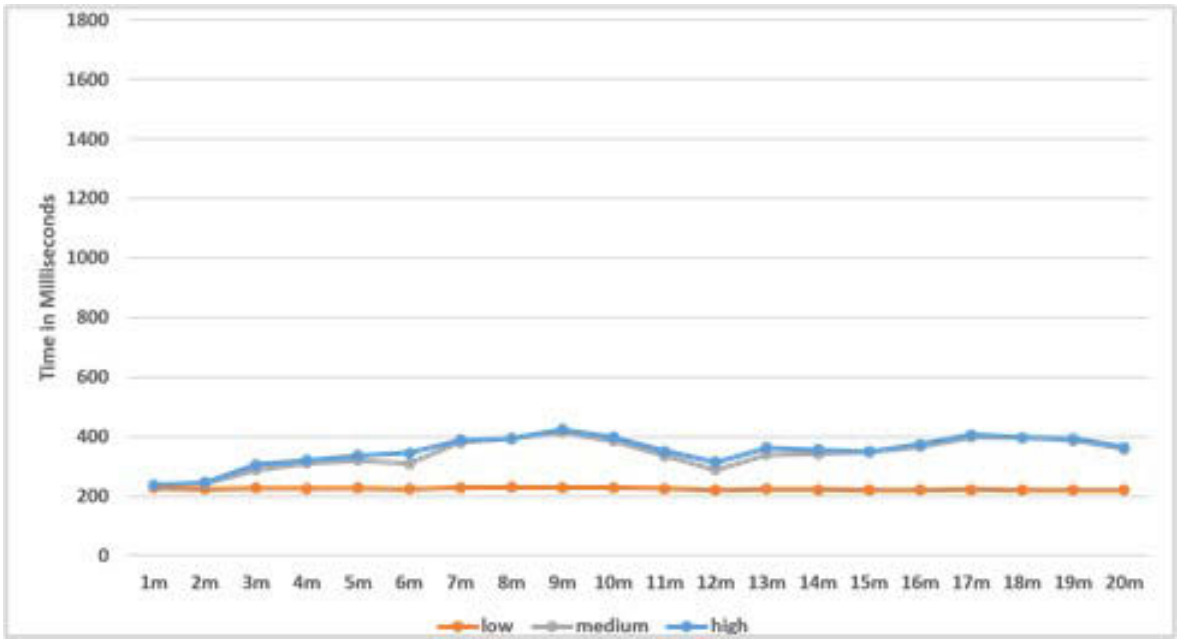



Figure 5.20: SMF Approach message delivery latency for 24 Nodes with a half frame length latency added.

Figure 5.19 to 5.24 show results for delivery latency in milliseconds. In the SMF results we added the average time needed to obtain a CB-Slot. For example, in the 24 nodes network, this delay is 192ms.

Figure 5.19 and 5.20 show that the latency is not much higher than this waiting time for low traffic loads. For higher loads, the latency grows due to temporary queues in the nodes. Figure 5.21, 5.22, 5.23, and 5.23, show the latency for larger networks and different traffic loads. The figures show that for medium and high traffic loads the latency increases at the end of the scenario likely indicating that the queues grow over time.

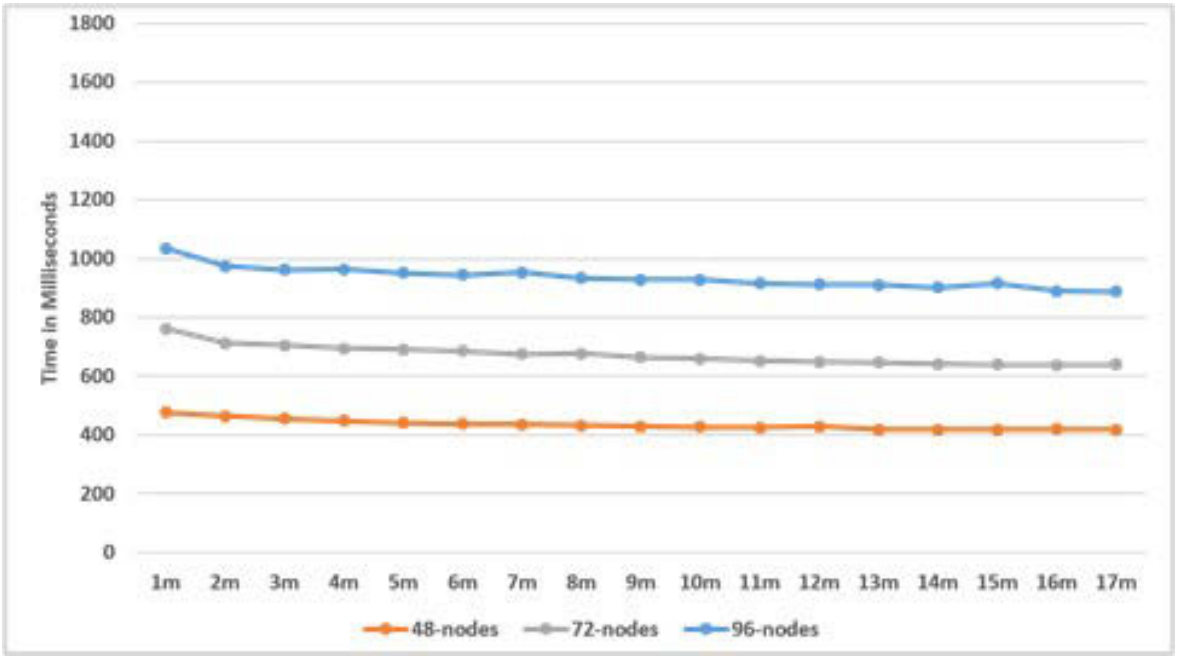



Figure 5.21: SMF Approach message delivery latency for low traffic load with a half frame length latency added.

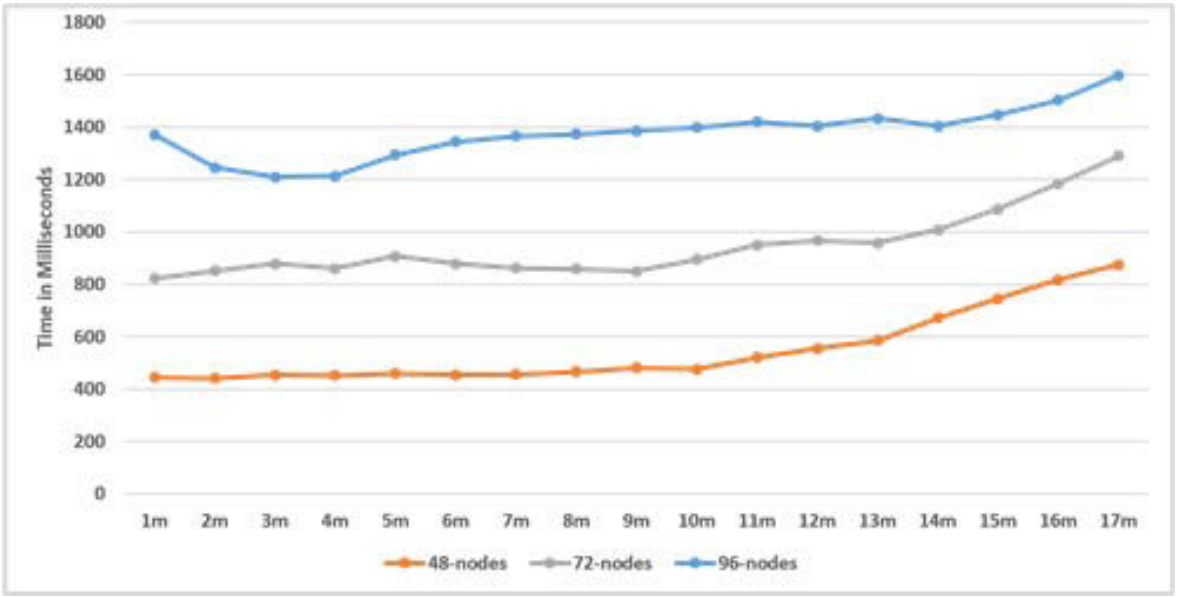



Figure 5.22: SMF Approach message delivery latency for medium traffic load with a half frame length latency added.

For the precomputed SCB experiments, we added the latency to obtain half a frame for up to four hops while for 5 to 8 hops, we added the latency to obtain 1 and a half frames. Figure 5.24 show latencies for the precomputed SCB and for different GCS.

The figures show that the latency in the PSCBT model does not depend on traffic load as long as no queue builds up in the ingress nodes. This is because PSCBT is implemented as a fixed schedule forcing each node to wait for its CB-Slot.

This makes the latency be the same for different traffic loads as long as the network does not get overloaded. Figure 5.24 shows that the latency to get half a frame is 96ms, while the latency for precomputed CB-Slot is

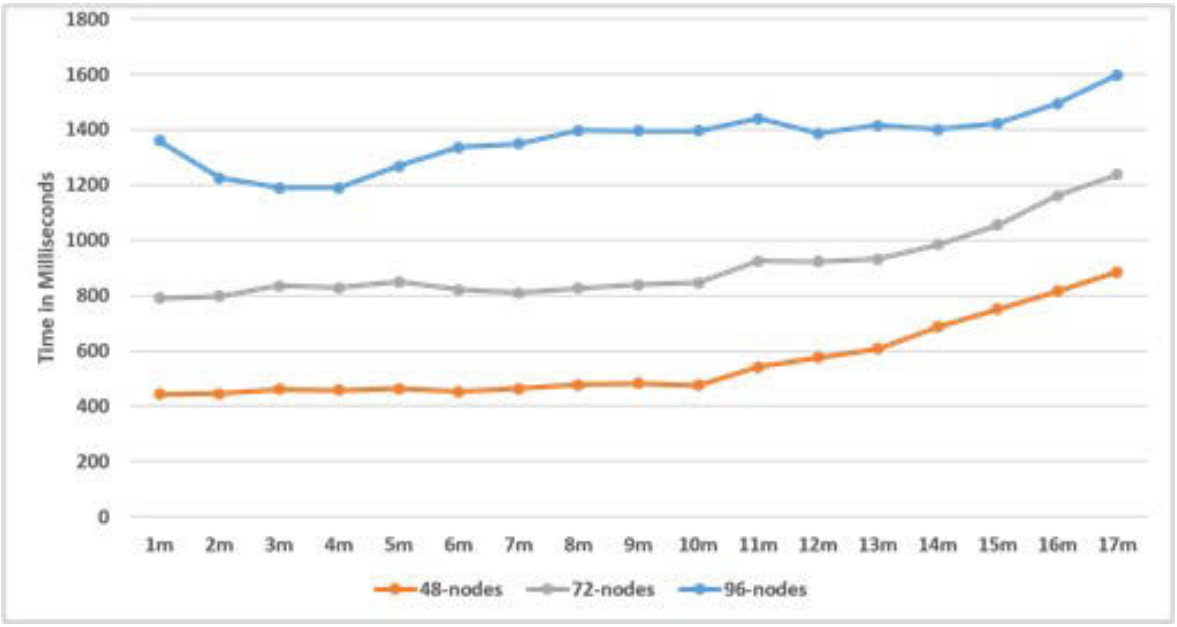



Figure 5.23: SMF Approach message delivery latency for high traffic load with a half frame length latency added.

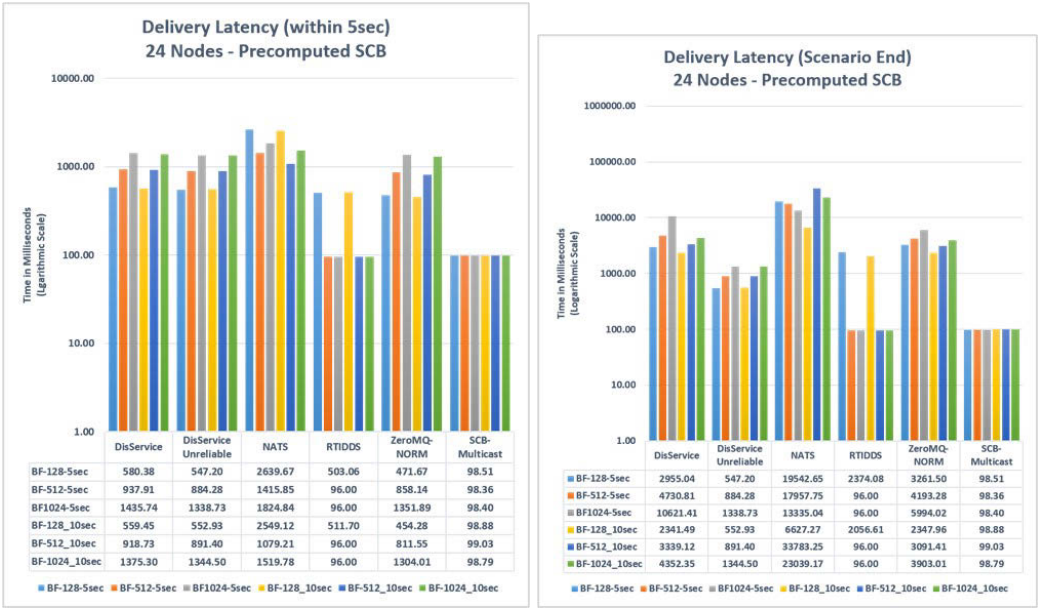
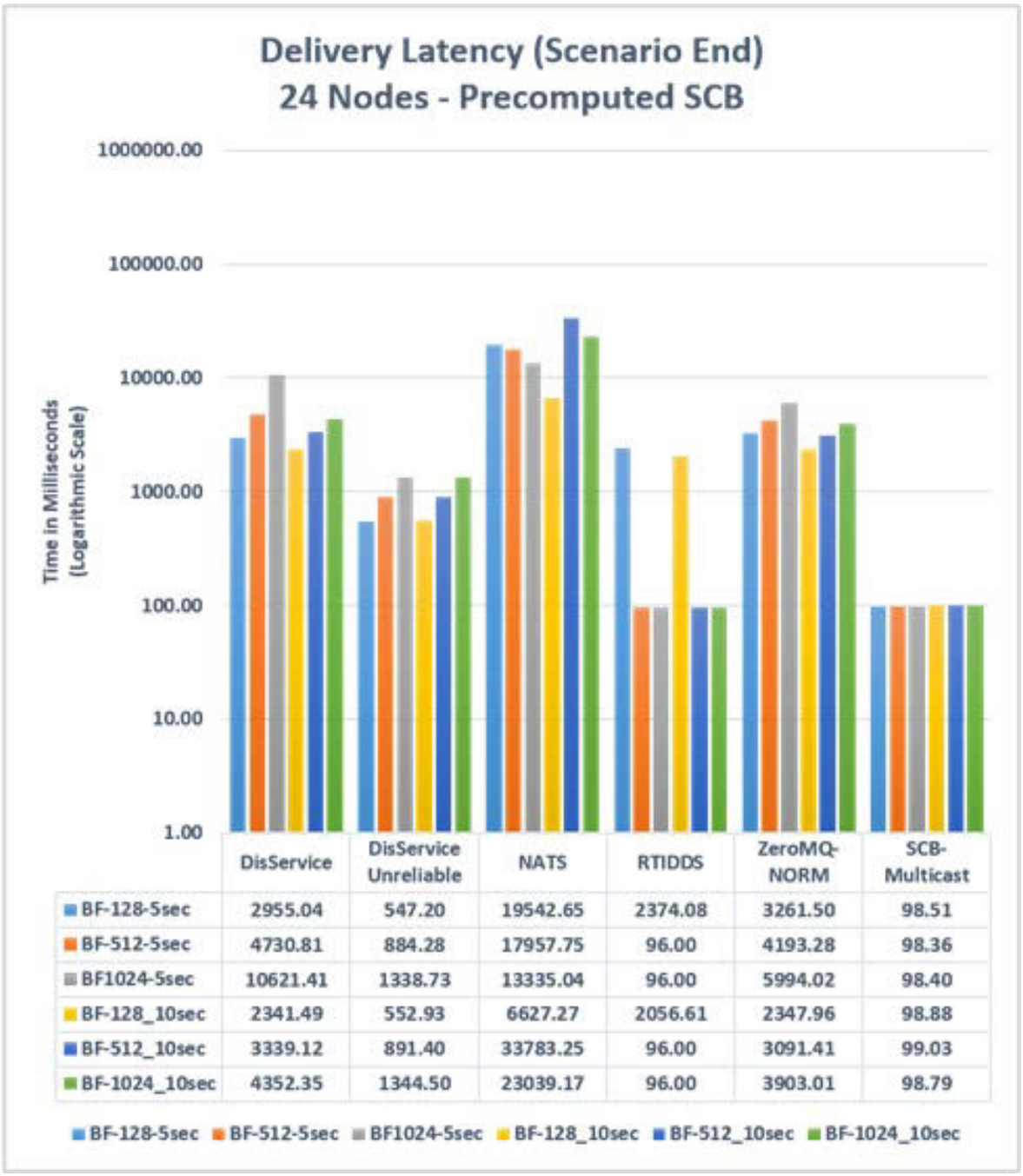



Figure 5.24: PSCB delivery latency for messages delivered within 5 seconds (left) and after 20 minutes (right) for 24 nodes.

slightly larger than 0.96ms. Only in a few cases, more than four hops are needed for the 24 nodes network.

### 5.10.3 Bandwidth Utilization




| | DisService | DisService Unreliable | NATS | RTIDDS | ZeroMQ-NORM | SCB-Multicast |
|---|---|---|---|---|---|---|
| BF-128-5sec | 2955.04 | 547.20 | 19542.65 | 2374.08 | 3261.50 | 98.51 |
| BF-512-5sec | 4730.81 | 884.28 | 17957.75 | 96.00 | 4193.28 | 98.36 |
| BF1024-5sec | 10621.41 | 1338.73 | 13335.04 | 96.00 | 5994.02 | 98.40 |
| BF-128_10sec | 2341.49 | 552.93 | 6627.27 | 2056.61 | 2347.96 | 98.88 |
| BF-512_10sec | 3339.12 | 891.40 | 33783.25 | 96.00 | 3091.41 | 99.03 |
| BF-1024_10sec | 4352.35 | 1344.50 | 23039.17 | 96.00 | 3903.01 | 98.79 |

Figure 5.25: PSCBT: Protocols bandwidth utilization with 24 Nodes.

Figure 5.25 and 5.26 show results for the bandwidth utilization of both the PSCBT and the SMF approach. The first figure shows results for the PSCBT in Kbps grouped by BFD size and frequency. Bandwidth utilization is fixed for PSCBT since each node only has one CB-Slot and limited throughput determined by the RFPipe data rate. In the simulation, we do not measure relay overhead, and therefore, since four slots are allocated, the real bandwidth (shown in Figure 5.25) is four times higher than what was measured. In practice, some of these slots may not be used but since they are allocated no other node can use them. The figure also shows that all GCS use substantially more bandwidth than simple IP Multicast over UDP.

Figure 5.26 shows the bandwidth utilization for the SMF approach, in particular, utilization grows with network

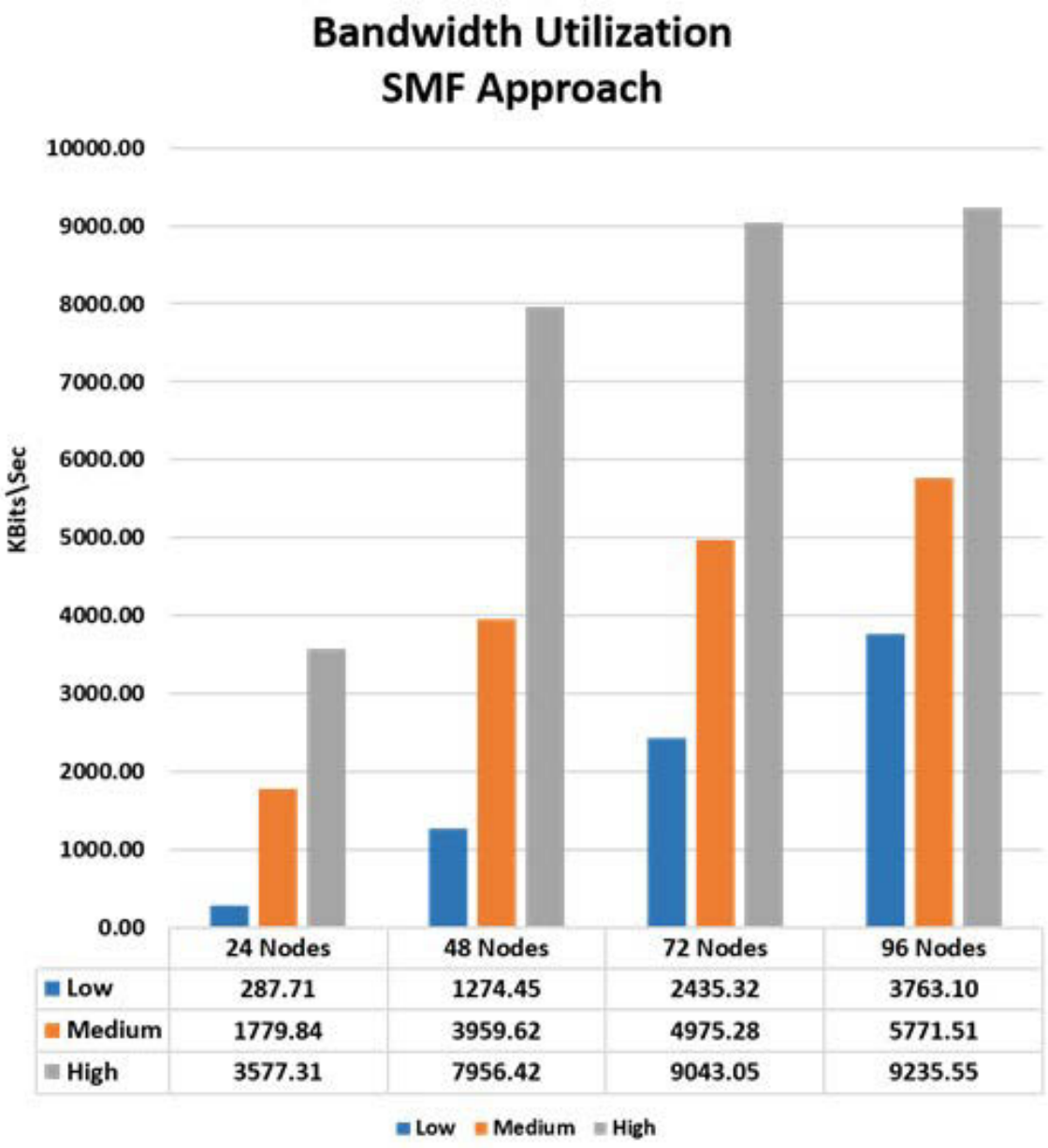



Figure 5.26: SMF: Bandwidth Utilization.

size and traffic load. Loads are substantially higher than in the PSCBT results because SMF executes full flooding at the UDP level whereas in PSCBT these retransmission are factored ahead of time in the connection map and translated into a simple ”on/off” flag for practical purposes.

## 5.11 Experiments: Scalability, PSCBT, and Jamming

In this set of experiments, we evaluated the scalability of several GCS and added the notion of adversarial jamming using the PSCBT model. For the experiments, we used from second 5000 to second 7000 of the Anglova scenario. Consistently with previous experiments, we set $L_{b,max}$ to 139 dB, bandwidth to 1.25 MHz, and link data rate to 875 kbps.

In particular, we performed the scalability experiment by evaluating the GCSs in configurations with 1, 2, 3,

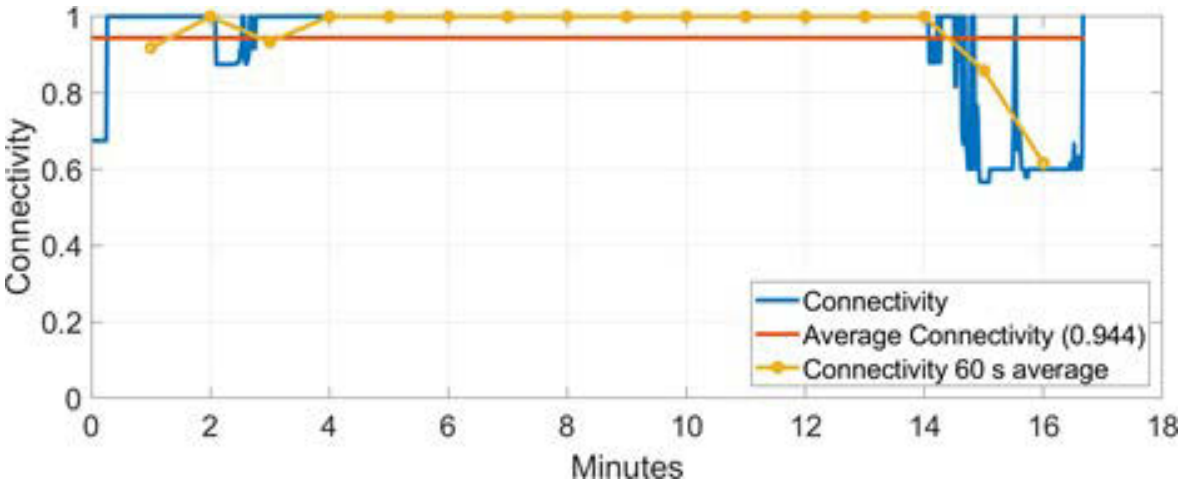



Figure 5.27: Connectivity of the overlay network.

and 4 companies of 24 nodes each. We configured each company with its own separated wideband network on a separate frequency and implemented inter-company connectivity as a separate wideband overlay network. Two nodes from each company acted as gateways between their company's network and the overlay network.

Figure 5.27 shows the connectivity of the overlay network for four companies that we extracted from the PSCBT, note that the connectivity is roughly the same when fewer companies are involved and that by connectivity we mean the fraction of node pairs that are connected via one or more hops in the SCB waveform.

### 5.11.1 Incorporating Adversarial Jamming Effects

In this set of experiments, we also considered a special case in which the company networks are jammed the whole time. We modeled the jamming as a 20dB reduction of $L_{b,"max"}$. The resulting fragmentation can be observed in Figure 5.28.

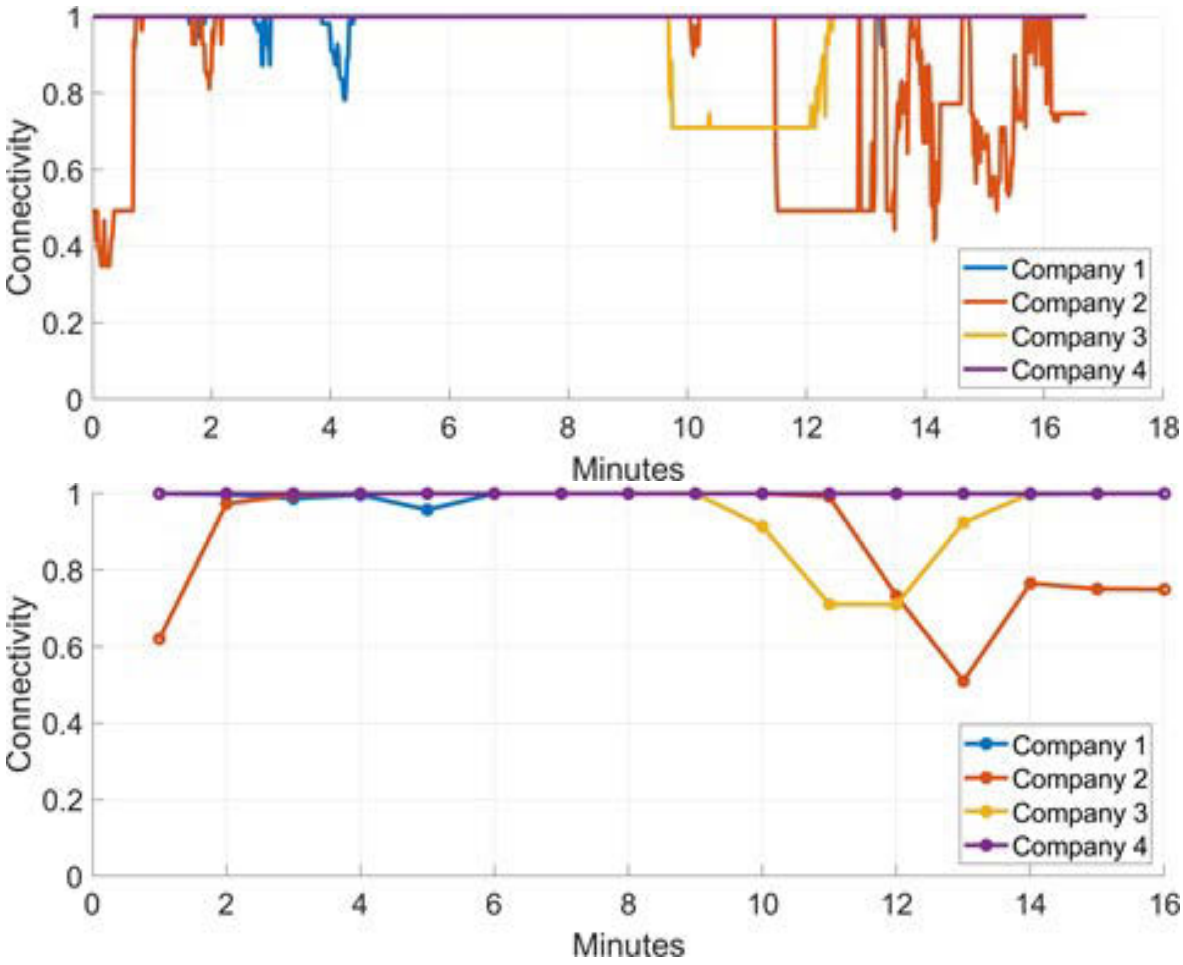



Figure 5.28: Connectivity of the company networks, the bottom plot consists of averages of 60 seconds.

### 5.11.2 Protocols Notes

Since this experiment involved multiple companies connected using an overlay network, we had to configure the GCSs to use two network interfaces at the same time.

For ZMQN, we relayed messages between the company and overlay interfaces based upon their source IP address. Because ZeroMQ SUB sockets did not provide a way to find the source address of an incoming message, we made senders add their IP addresses (4 bytes) to each message sent as an additional header. To prevent messages from looping, we added a method to drop duplicate messages that were received within a preset time threshold. Note that this dropping occurs at the application level, and dropping messages that have an identical payload may not have the intended behavior in experiments different from the ones shown in this section. To use our implementation in different experiments, it is necessary to provide a custom way of differentiating between messages that share the same payload. In our case, all the messages generated by the Driver are unique.

For GDEM, we introduced a relay function that sits between the GDEM and SUSP layers and relays messages between the company and overlay networks. Similarly to DS, we also tested GDEM in two different configurations. The first is an unreliable configuration with no repair window configured to not recover missing messages. The second is a reliable configuration where the GDEM repair window is set to 10 (which will make

GDEM attempt to repair missing messages that are still within the repair window).

### 5.11.3 Experiment setup

For this set of experiments, we prepared configurations of 24, 48, 72, and 96 nodes. In the 24 nodes configuration, we included all nodes of Company 1, in the 48 the ones from Company 1 and 3, in the 72 nodes configuration all nodes in Company 1, 3, and 4. Finally, in the 96 nodes, we include all four companies.

In each company, we configured the first two nodes with two network interfaces. The GCSs used the first interface to communicate with local nodes and the second with other relay nodes. Based on the number of companies involved in an experiment, the overlay network had either 0, 4, 6, or 8 nodes in total.

Within each company, we started by considering that with a total bandwidth of 875 kbps and assuming a static TDMA schedule with equal bandwidth allocation, each node would on average have a bandwidth of 9.1 kbps. However, given that the first two nodes of each company were also part of the overlay network and had to relay data between the local company and the rest of the companies, we adjusted the TDMA schedule to give more slots to the first two nodes. In particular, we gave the first two nodes 29 kbps and each of the other nodes in the company 7.1 kbps.

In the overlay network, the total bandwidth was divided equally between the number of nodes that participated in the overlay. For this reason, depending on the number of companies (and consequently the number of nodes in the overlay network) the bandwidth was respectively 55 kbps with two companies, 36 kbps with 3, and 27 kbps with 4.

Ideally, we would have used EMANE's TDMA MAC Layer for this experimentation but unfortunately, we observed that it was very sensitive to time differences between VMs deployed over multiple hypervisors. The significant error introduced by that sensitivity forced us to rely on the EMANE's CommEffect (CE) model instead.

CE allows adding network effects such as delay, loss, jitter, and bandwidth limits to the ingress side of a network interface. Typically, this mechanism works well but not in our scenario. In particular, when there are few sending nodes but many receiving nodes, the firsts may emit more data than what should be theoretically possible. This happens because the bandwidth is spread across the receivers rather than senders. This fails to emulate sending via a TDMA waveform, as it allows a sender to transmit more data than what is allowed by its slot.

To compensate for this phenomenon, we used the Linux kernel Traffic Control to employ a token bucket filter that prevented the nodes from exceeding the transmission limit of SCB. The token bucket mechanism works by having several tokens that can be spent on sending data. The tokens are filled up at a set rate until the bucket is full again. Packets are only sent if sufficient tokes are available and they are otherwise queued.

In the following, two types of comparison are proposed. The first comparison examines the cross-protocol performance within the same configuration (e.g., three companies, with a 10-seconds cut-off time). The second comparison looks at the scalability within the same protocol as the number of nodes increases. Note that for DS and GDEM, we report two sets of results, one for when they are configured to be reliable (DS-R and GDEM-R),

and one for when they are configured to be unreliable (DS-U and GDEM-U).

### 5.11.4 Delivery ratio results

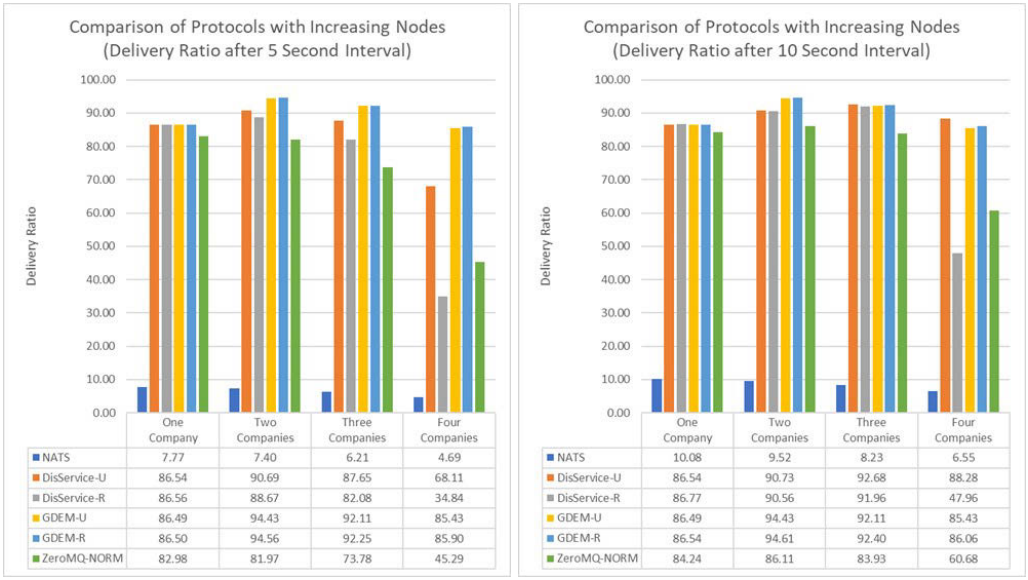



Figure 5.29: Delivery ratios of different protocols with 5 (left) and 10 (right) seconds cutoff for multiple company sizes.

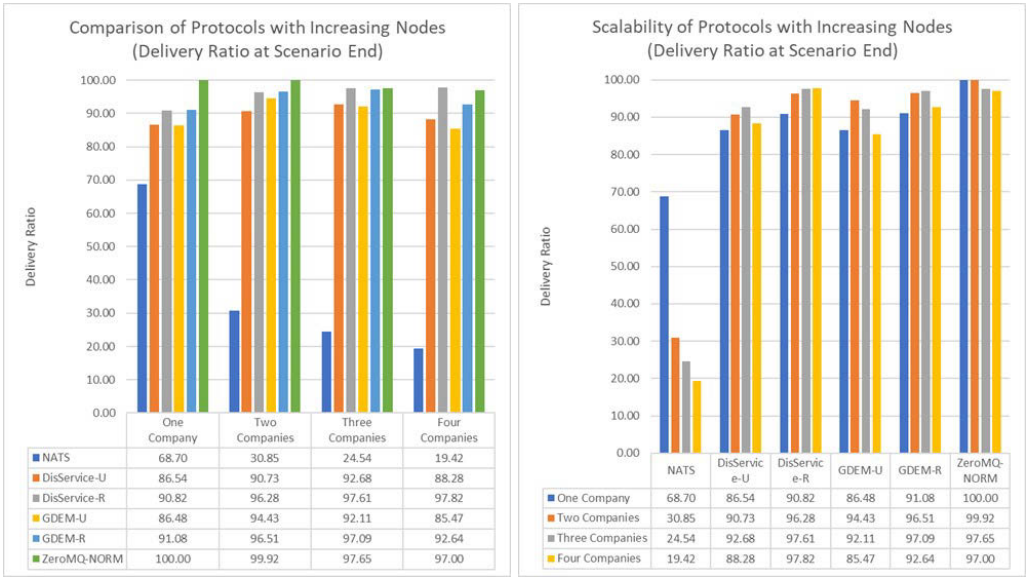


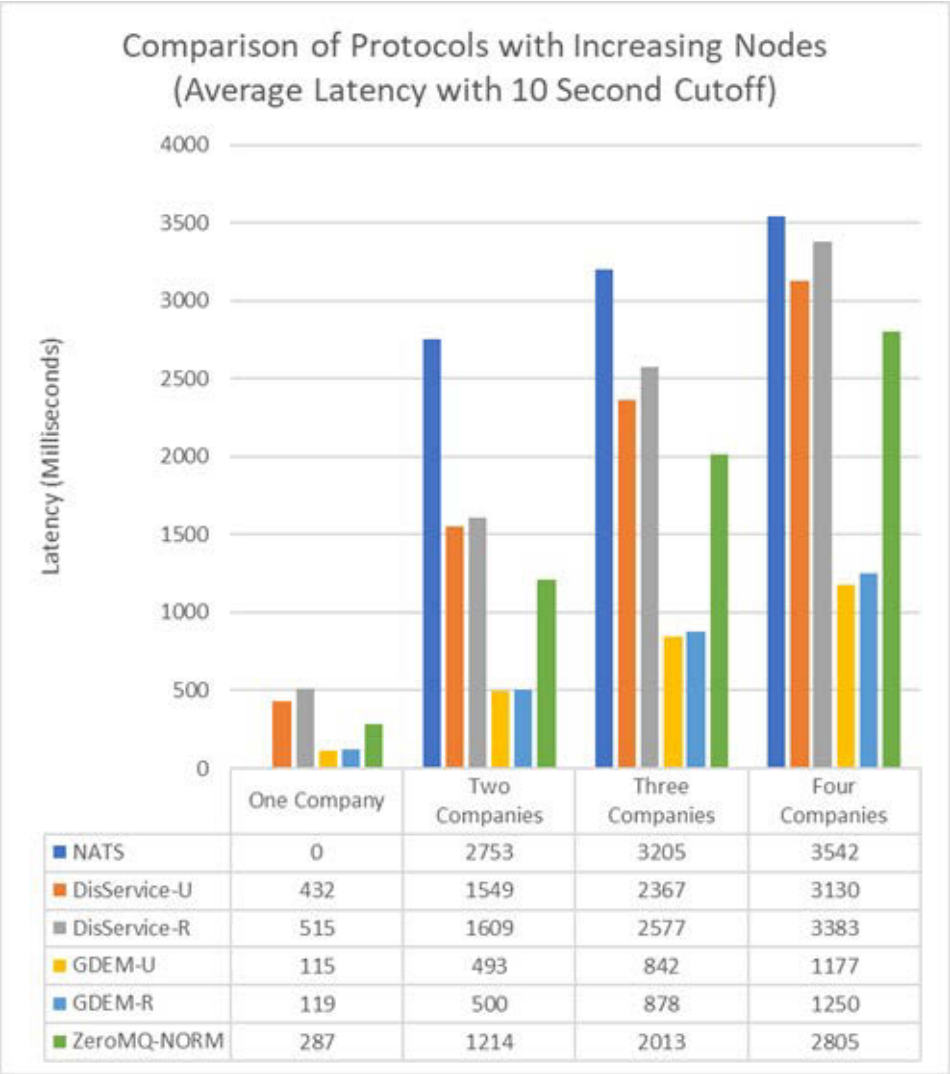

Figure 5.30: Delivery ratios with no cutoff times.

Figure 5.29 and 5.30, show the Delivery Ratio at different cutoffs. Figure 5.29 shows that the 5 seconds cut-off results in a very low delivery ratio for NATS in any topology. This is an expected result from a unicast protocol deployed in a bandwidth-constrained environment. While the other protocols perform roughly the same in this analysis, GDEM seems to show comparatively less degradation at the increase of the number of companies involved in the experiment. The Figure also shows results for the 10-second cutoff. NATS is still the worst performer. DS and GDEM present the best performance. ZMQN continues to perform slightly worse than the other two. Figure 5.30 shows results with no cutoff. In practice, this is the delivery ratio at the end of the scenario (i.e., after 2000 seconds). From the figure, it is possible to see a decisive improvement in all the

reliable protocols since they had more time to deliver messages. ZMQN slightly outperforms all other reliable protocols for one and two companies, and DS slightly outperforms them with four companies.

Figure 5.30 also shows the same data clustered by protocol highlighting scalability as the number of nodes increases. DS and GDEM show improvements when scaling from 1 to 3 companies. DS-R shows the best performance with four companies while ZMQN performance decreases slightly as the number of companies grows. NATS performance can be seen decreasing as expected.

### 5.11.5 Latency results




| | One Company | Two Companies | Three Companies | Four Companies |
|---|---|---|---|---|
| NATS | 0 | 2753 | 3205 | 3542 |
| DisService-U | 432 | 1549 | 2367 | 3130 |
| DisService-R | 515 | 1609 | 2577 | 3383 |
| GDEM-U | 115 | 493 | 842 | 1177 |
| GDEM-R | 119 | 500 | 878 | 1250 |
| ZeroMQ-NORM | 287 | 1214 | 2013 | 2805 |

Figure 5.31: Comparison of the Latency of different protocols with a 10 second cutoff and different company sizes.

Figure 5.31 shows the average message delivery latency for each protocol, with an increasing number of nodes with a cut-off of 10 seconds. While some messages take ten or more seconds to arrive, the averages never exceed 4 seconds. From the picture, it is possible to see that there is no configuration in which NATS can deliver any message on average in less than 10 seconds. ZMQN and DS latencies are comparable while GDEM presents better latencies while achieving comparable delivery rates.

### 5.11.6 Bandwidth results

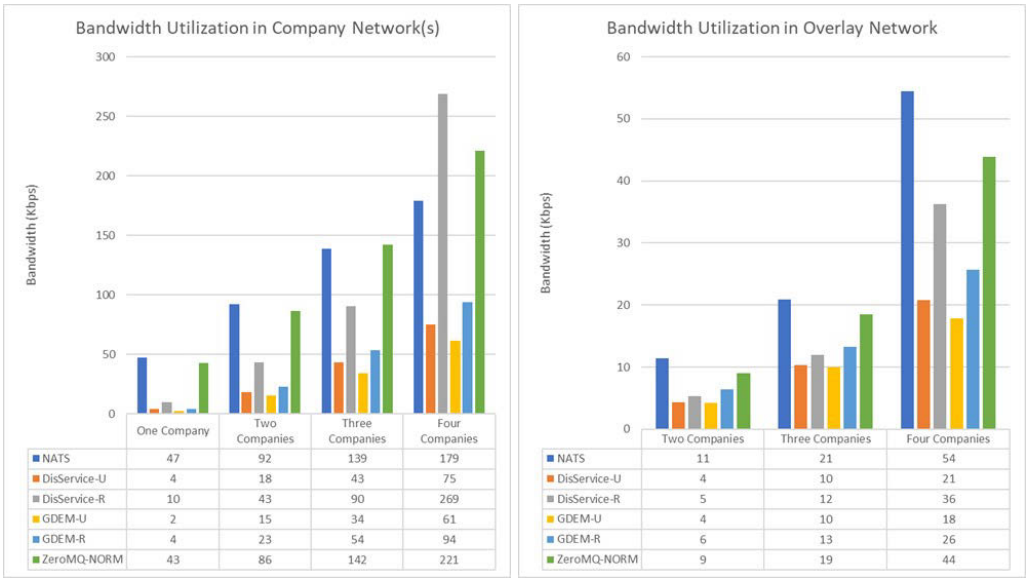
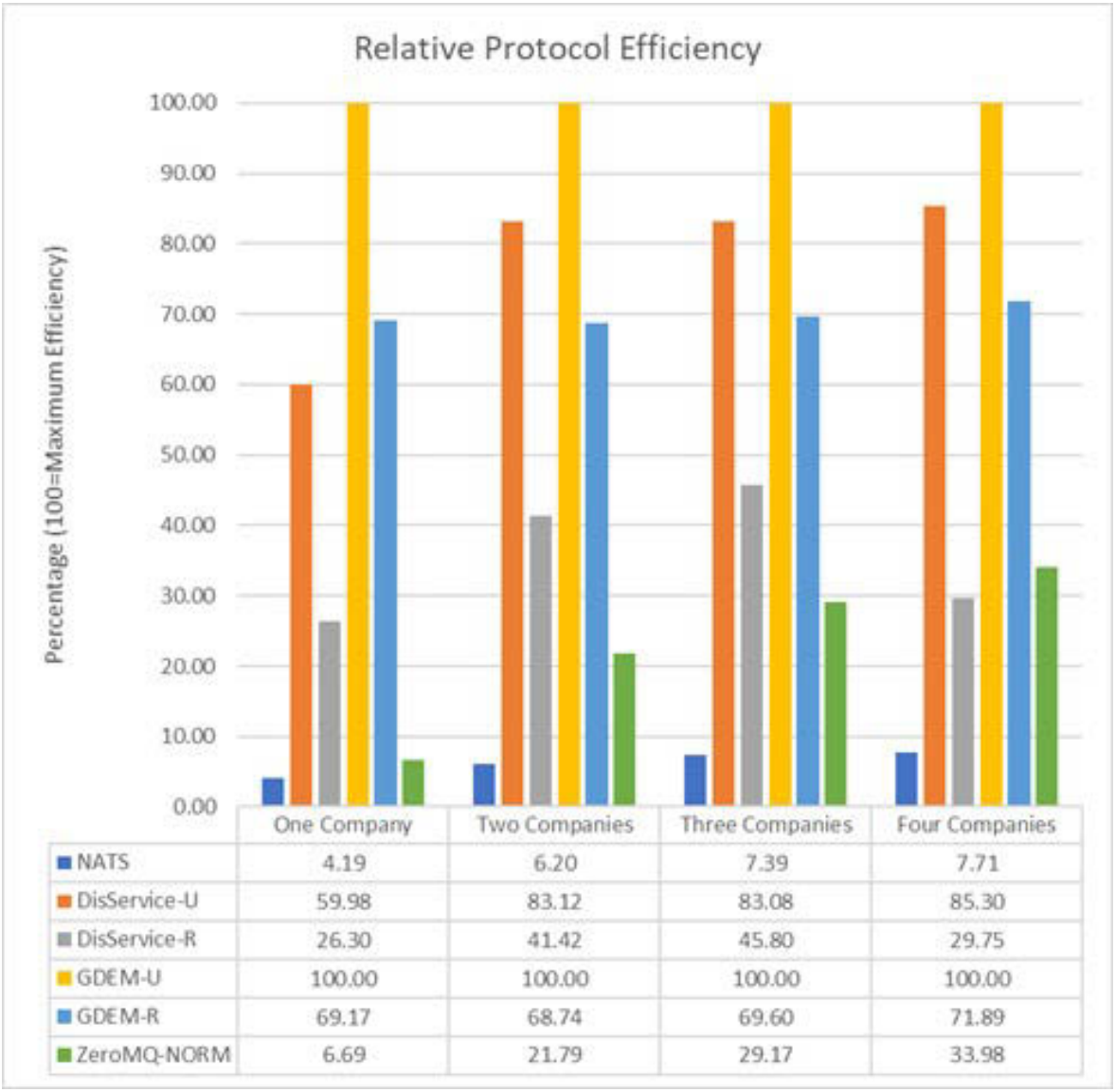



Figure 5.32: Bandwidth utilization within company networks (left) and in the overlay (right).

Figure 5.32 shows bandwidth utilization within companies as well as in the overlay network. As expected, NATS is the protocol with the highest levels of bandwidth usage, followed by ZMQN. Surprisingly as the number of companies increases ZMQN overtakes NATS as the most bandwidth-intensive protocol. It is worth remembering that NATS still presents worse performance in terms of delivery rates. The two GDEM flavors perform better than the respective DS ones. While GDEM offers reliability through a repair window (set to 10 in these experiments), DS-R attempts to repair all missing traffic. This results in a small increase in the delivery ratio at a significant bandwidth cost. Note that the bandwidth was reported as the sum of the sent traffic in each company. Figure 5.32 also shows the bandwidth utilization in the overlay network. Again NATS is the worst performer. DS performs significantly worse in the four company deployment.

Finally, Figure 5.33 shows the relative efficiency of the protocols, computed as a function of the overall volume of data delivered versus the bandwidth utilization, with the best performing protocol scaled to 100. We computed Protocol Efficiency by taking the total volume of useful data delivered versus the overall network utilization. The best protocol was then set to be 100 %, and the relative efficiency of the other protocols was calculated as a relation to the best performing protocol. From the figure, it is possible to see that GDEM-U was the best performer. With one company, GDEM-R was the second-best performer with DS-U being the third best.

On the other hand, with two, three, and four companies, DS-U performed a little better than GDEM-R. Note that ”best” performer, in this case, does not mean highest delivery ratio (which was typically achieved by GDEM-R, DS-R or ZMQN), but has to be intended as a trade-off measure taking into account the network utilization and the delivery ratio that was achieved for that network utilization.




| | One Company | Two Companies | Three Companies | Four Companies |
|---|---|---|---|---|
| NATS | 4.19 | 6.20 | 7.39 | 7.71 |
| DisService-U | 59.98 | 83.12 | 83.08 | 85.30 |
| DisService-R | 26.30 | 41.42 | 45.80 | 29.75 |
| GDEM-U | 100.00 | 100.00 | 100.00 | 100.00 |
| GDEM-R | 69.17 | 68.74 | 69.60 | 71.89 |
| ZeroMQ-NORM | 6.69 | 21.79 | 29.17 | 33.98 |

Figure 5.33: Overall Relative Efficiency of the Group Communication Protocols.

# Chapter 6

# Network Monitoring and Adaptation in Constrained Environments

The extreme dynamism and heterogeneity of networks used to support TDR operations make it impossible to optimally tune the behavior of communications middleware through predefined configurations or simple heuristic solutions. Moreover, the nodes that use these networks are highly diverse and range from constrained sensors and other forms of battery-operated portable devices to powerful servers mounted on vehicles, ships, and aircraft.

Communication within TDR networks is challenging and easily disrupted, missions are often on the move, and the network infrastructure is not static but composed of a combination of wireless sensors networks and MANETs easily disrupted by physical obstacles and interference.

To support end-user applications, guarantee system performance, and inter-operability in such constrained environments, it is critical to adapt the volume and type of traffic generated by applications to the continuously varying network conditions.

These factors call for network-aware solutions that can continuously re-tune themselves to adapt their behavior to the ever-changing operating conditions. Moreover, these solutions must be capable of collecting network status information, generating actionable knowledge (including load, link types, quality, and network topology), and delivering this information where it can be processed and used despite the constrained network infrastructure.

To this end, we developed SENSEI, a set of components specifically designed to provide efficient network state detection and adaptation in constrained networks. SENSEI's services collect data on resource availability and usage statistics, efficiently distribute this information over the constrained links, increase network observability by enabling operators to monitor the state of the network, and control and adapt other components. In particular, SENSEI can cooperate with many services of the Agile Communications Middleware (ACM) to improve the use of network resources and show network information through appropriately designed graphical interfaces.

SENSEI is a distributed and extendable microservice-based solution that can increase network observability and improve the use of available network resources.

We designed SENSEI following the microservice architectural pattern [110], which enables the development of loosely coupled, specialized, and independent services that can interoperate through well-defined interfaces.

The small scope of each microservice expedites development by reducing interaction between team members; intrinsic modularity makes these services easier to replace, evolve, and maintain. Moreover, microservice architectures enable graceful degradation by keeping each service separated at the process level, consequently barring the failure of a single service from jeopardizing the whole group.

The rest of this chapter is organized as follows. The first and second sections respectively introduce SENSEI and its architecture. The third section discusses SENSEI's smart status exchange algorithms. The fourth section presents SENSEI's passive bandwidth estimation algorithm. The fifth and sixth sections discuss respectively complexities in deploying SENSEI and NetCacher, a component of the ACM that we used to evaluate adaptation to network conditions during video-streaming. Section 6.6 to 6.9 present several experiments evaluating SENSEI's performance and algorithms in a variety of conditions.

## 6.1 Architecture

SENSEI's services divide logically into four layers. Services belonging to the Sensor Layer monitor and extract information from network devices and generate self-contained reports regarding characteristics such as running services, available resources, and exchanged traffic.

Services belonging to the Aggregation and Share Layer receive the reports generated by the Sensor Layer, perform some data-agnostic aggregation over them, and efficiently distribute the resulting information over the network. Finally, services belonging to the Process Layer consume other layers' messages to provide non-agnostic aggregation and actionable information. It is worth noting that these services are not isolated from each other. For example, the Share Layer can use results produced by the Processing Layer to improve delivery policies, such as only sending relevant information to interested neighbors.

Communication within a single SENSEI instance is done through the NATS messaging bus, while inter-instance communication uses a specialized microservice called Smart Network Status Exchange Service (SNSE). This choice was driven by two motivations. First, NATS relies on TCP for message delivery, and many studies have shown that TCP performs poorly in degraded and wireless networks. Second, by limiting inter-service communication to processes running on the same host, we can reduce the negative impact that the increased communication overhead of the microservice architecture could have on already bandwidth-constrained links.

SENSEI encodes information into a protobuf encoded format called measure. Each measure contains a subject, a string identifying the type of measure, a timestamp, used to signal when the measure was generated, and three maps, one for string values, one for integers, and one for doubles. Each measure can be instantaneous or cumulative depending on whether it contains values over a time interval or represents an event at a specific timestamp.

### 6.1.1 Passive and Active network monitoring

Monitoring solutions can be active or passive depending on whether or not they introduce probing traffic in a network to perform analysis operations. Active approaches are generally more accurate. They can provide information on demand but suffer from the side-effect of changing the normal state of the network. The probing traffic used to analyze the network stack and notify other applications of the results can compound negatively on the state of the network, degrading other transmissions. For example, a common approach to measure the available bandwidth is to saturate the link between two hosts with mock traffic and log the receiver's rate. This saturation can prevent other traffic from circulating the same network.

On the other hand, passive monitoring only observes traffic already in the network. Common approaches for passive monitoring are to deploy passive sensors in selected nodes and capture network traffic or rely on reports from network devices.

Unfortunately, information that can be obtained via passive monitoring is limited to active end-to-end communications and is generally less accurate than active methods. Considering the specific limitations of networks used to support TDR operations, we designed SENSEI to primarily be based on passive monitoring and only uses active mechanisms when specifically requested by network operators.

### 6.1.2 NetSensor

NetSensor is the primary service of the Sensor layer. We designed this component to passively monitor traffic between hosts to extract characterizing information such as the average bytes and packets exchanged by applications and RTT between hosts. In particular, NetSensor uses the Pcap/WinPcap library to capture network packets, build statistics on network usage, discover and identify local gateways, enable bandwidth estimation, and assess the latency between hosts by processing ICMP or TCP's (S)ACK/NAK packets.

The ICMP latency detection mechanism calculates latency by measuring the delta between Echo Request and Replies, while the TCP mechanism computes the delta between packets flagged as PSH and relevant acknowledgments.

NetSensor implements two simple local topology detection algorithms. The first identifies network gateways by looking for nodes with a disproportionate number of IP addresses associated with a single MAC address. The second algorithm classifies as local nodes, all the nodes that have a MAC address different from one that belongs to a gateway.

Finally, NetSensor can also be configured to group addresses that belong to specific network segments to overcome tricky configurations that would fail to be recognized by the previous algorithms.

### 6.1.3 Node Monitor

The NodeMonitor is a component that spans the Sensor and Aggregation layers. Its primary task is that of collecting reports from the sensor layer and converting to SENSEI's measure format any report encoded differently. Figure 6.1 shows the NodeMonitor receiving statistics generated by Mockets and NetSensor, in particular, Mockets used to use a custom report format that had to be converted into SENSEI's format before it could be

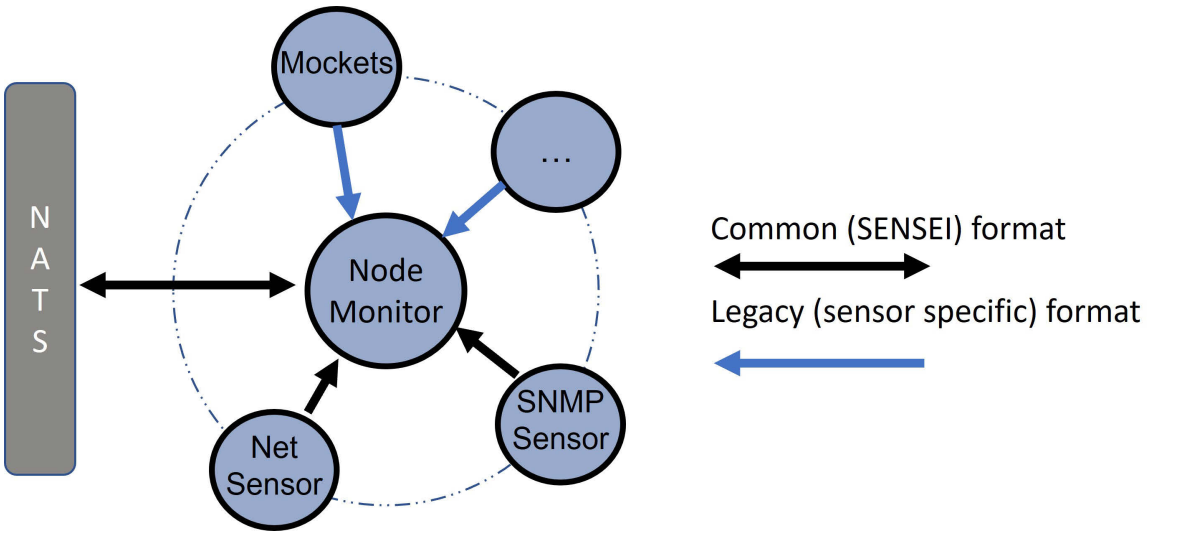



Figure 6.1: SENSEI's sensor aggregation service harvesting data from a number of sensors.

made available to other services. The SNMPSensor in the figure is another service we developed to integrate SNMP information in SENSEI based on the SNMP4j [111] library.

The NodeMonitor also directly harvests a variety of information from the localhost, including CPU, memory, battery, and OS-related information through the SIGAR [112] library which provides a cross-platform interface to information on computer hardware and operating system activity.

### 6.1.4 SNSE

SNSE is the primary service of the aggregation and sharing layers. This service performs data-agnostic aggregation and distributes information over constrained links.

It also collaborates with other services from the processing layer to provide content- and context-aware information distribution of network state information, which uses characteristics of neighboring nodes to significantly reduce state sharing overhead.

### 6.1.5 NetSupervisor

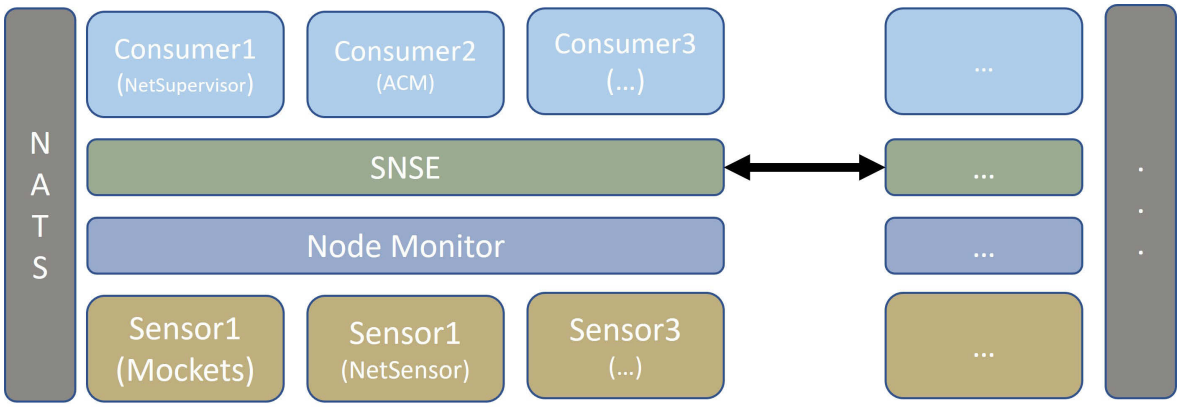



Figure 6.2: SENSEI Architecture: interaction between Sensors, SENSEI's services, and the NATS messaging system.

NetSupervisor is the primary service of the Processing layer. We designed this component to process information obtained from remote and local nodes to enable content-aware aggregation and produce actionable results.

Figure 6.2 shows SENSEI architecture up to this point. On the bottom, services in the Sensor layer harvest information about the status of the network which is then converted if necessary into SENSEI's measure format and made available to other services locally through NATS and remotely by SNSE.

Upon receiving network statistics, NetSupervisor processes that information and generates reports regarding the network status. In particular, NetSupervisor computes a local summary for each cluster containing exchanged traffic, latency, connectivity, and packet loss. Moreover, NetSupervisor creates a link report for each pair of clusters. This report describes the characteristics of each connection in terms of bandwidth, latency, and packet loss.

NetSupervisor also implements two other services: The Alert and Election algorithms. The Alert algorithm tracks unique identifiable measures and calls a given callback in response to the verification of an event. If no callback is specified, it generates a message that reports on the anomaly. There are currently three possible triggers: a variation of a tracked value from a certain quantity, a percent variation between two consecutive values, or equivalence with a target value. It is also possible to specify if the reference has to change each time a new alert is triggered or if the first value detected will be a permanent reference.

The Election, or Link Detection, algorithm, infers the baseline characteristics of an unknown link. The inference process is purely heuristic and operates as follows. First, a subject matter expert writes a Reference Link Status for each link type containing several metrics that characterize each link, such as expected maximum throughput, minimum latency, and packet loss. The expert then assigns to each value a score, reflecting how likely the link under consideration will have that characteristic.

Subsequently, the run-time characteristics of the link are compared with values in the reference files, and the description that better fits the observed attributes is chosen.

In a typical scenario, this algorithm can be used to identify links that change communication technology, such as switching from SATCOM to LTE or UHF radio, and to differentiate between different states of the same link, e.g., good vs saturated.

Table 6.1 shows the thresholds for which SENSEI computes scores after analyzing the reference files. In the figure, ET and EL are the expected throughput and latency, whereas T and L are the values of throughput and latency detected at run-time. During the Link Analysis phase, NetSupervisor will use the reference values specified in the configuration files to assign a score to each configured link type. Each score will be the sum of the sub-scores obtained for latency and throughput values extrapolated in turn by comparing the value obtained at run-time with the associated reference table mentioned earlier.

Finally, NetSupervisor selects the link type with the higher score as the assumed link type.

To clarify, if there are two link types, A and B, and at run time we detect a throughput of value T and latency L, the score of A can be obtained by summing the relative scores of latency and throughput affinity obtained evaluating the respective tables.

| Throughput Thresholds | Latency Thresholds |
|---|---|
| $T >= ET$ | $0.8 * EL <= L < 1.2 * EL$ |
| $0.9 * ET <= T < ET$ | $1.2 * EL <= L < 0.2 * EL$ |
| $0.75 * ET <= T < ET$ | $L > 2 * EL$ |
| $0.50 * ET <= T < ET$ | $0.5 * EL <= L < 0.8 * EL$ |
| $0.25 * ET <= T < ET$ | $0.2 * EL <= L < 0.5 * EL$ |
| $0.1 * ET <= T < ET$ | $0.1 * EL <= L < 0.2 * EL$ |
| $T < 0.1 * ET$ | $L < 0.1 * EL$ |

Table 6.1: Throughput and Latency Thresholds tables.

The scores can also be considered a confidence index, the larger the difference between each score value, the safer the classification of the link. For argument's sake, if after the extrapolation the total score of A is higher than the total score of B, NetSupervisor will infer that the link type is A and vice versa.

NetSupervisor also supports a simplified version of the Election Algorithm where a subject expert has to only fill reference values for bandwidth and latency. NetSupervisor will then compare the distance between the values detected at run-time and the reference values and pick the link with the smallest distance.

### 6.1.6 Visualization

A critical component of situation awareness and mission success is the timely delivery of information between nodes. Since network performance and hardware status can have a great impact on node communication, operators can gain great insight by observing baseline measurements of network performance and other characterizing information such as which applications and devices are generating anomalous amounts of traffic.

To satisfy this need, we integrated SENSEI in 3 visualization services.

NetViewer is a user interface designed for network administrators that supplies a graphical environment to provide access to SENSEI information. In particular, it can show the aggregated network topology, traffic over time, latencies, packet loss, and so on. NetViewer also supplies facilities to request more information from SENSEI to enable drilling down on information not periodically exchanged between instances. NetViewer also shows live updates on nodes joining and leaving the network, hardware, software, and network info such as incoming and outgoing traffic rates for each node, and statistics of other middleware such as DisService and Mockets. Figure 6.3 shows NetViewer. In particular, the figure shows an aggregated topology on top, and a chart of traffic over time right below.

Mirage is a tool designed to be utilized by operators directly into the field and capable of displaying SENSEI's as a layer on top of other information. Since network administrators are not the primary targets of this component, the information that it shows is greatly simplified and limited to reporting about the connection status between clusters of nodes. We designed Mirage to aggregate information coming from various sources and display what results on top of a map. Figure 6.4 shows SENSEI information displayed on top of a map. In the figure, it is possible to see links of various colors, each color representing a different link condition that can be used to gain insight into the status of the network at a glance. The figure also shows several little icons representing positional updates of other units or sensor reports.

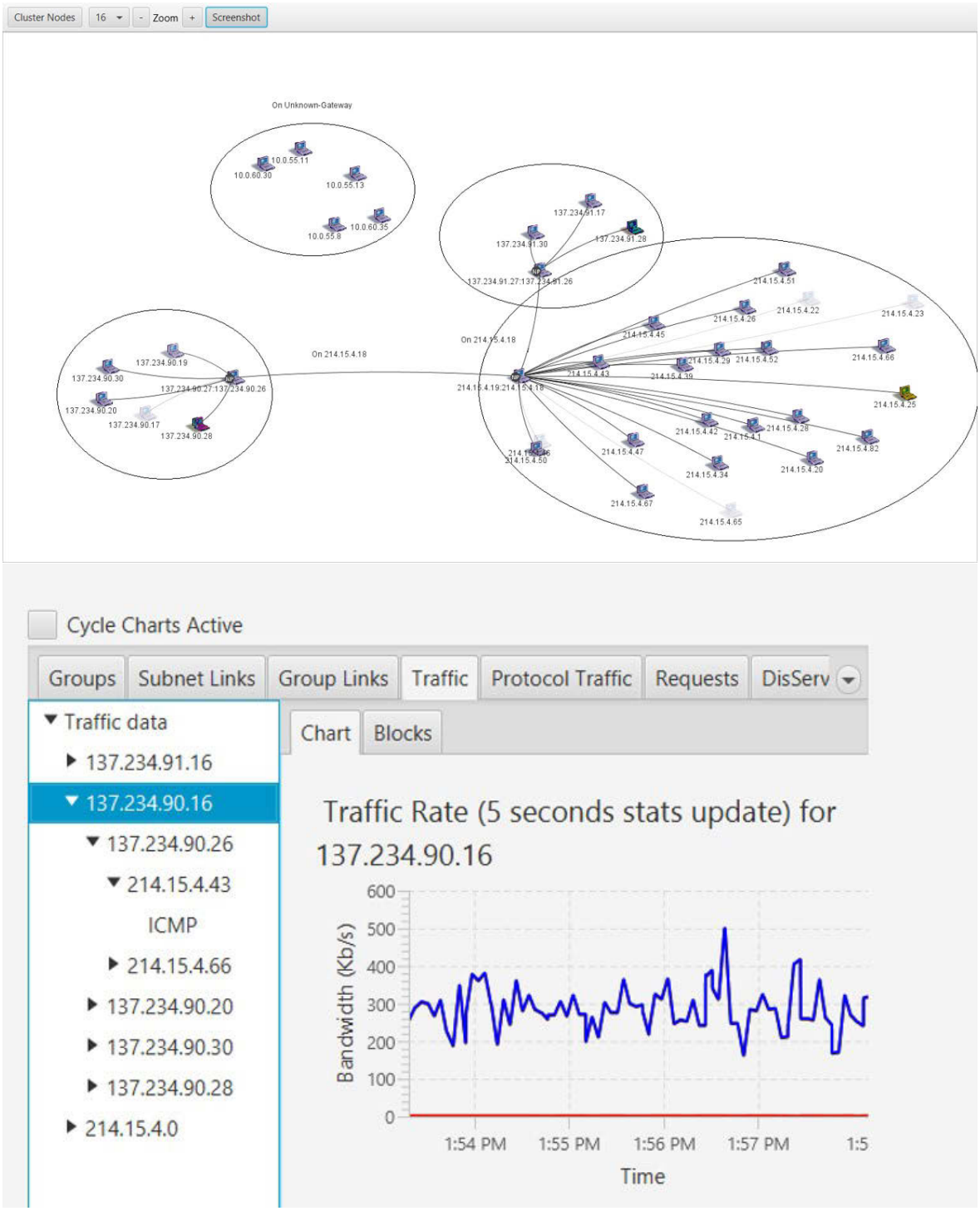



Figure 6.3: NetViewer user interface.

We designed SENSEI to easily integrate with other services. In particular, SENSEI encodes information in stateless protobuf messages periodically sent and updated when new information is available. Figure 6.5 shows SENSEI integration in another application used by the armed forces called KilSwitch.

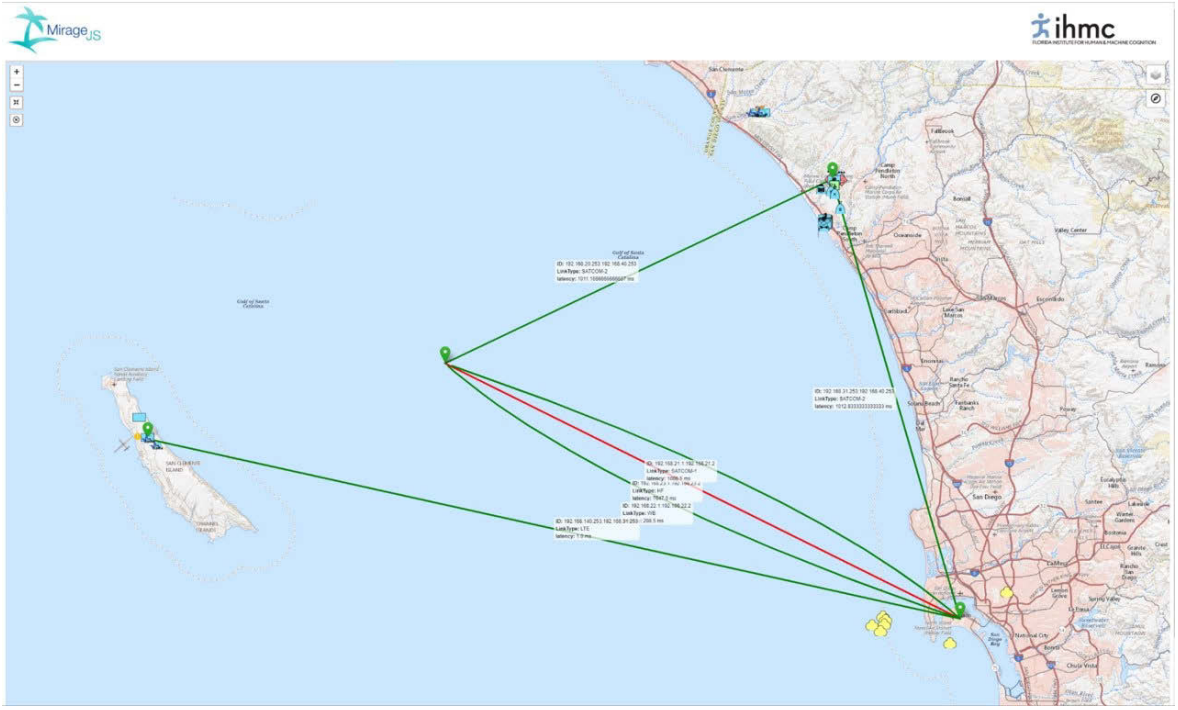



Figure 6.4: Mirage user interface.

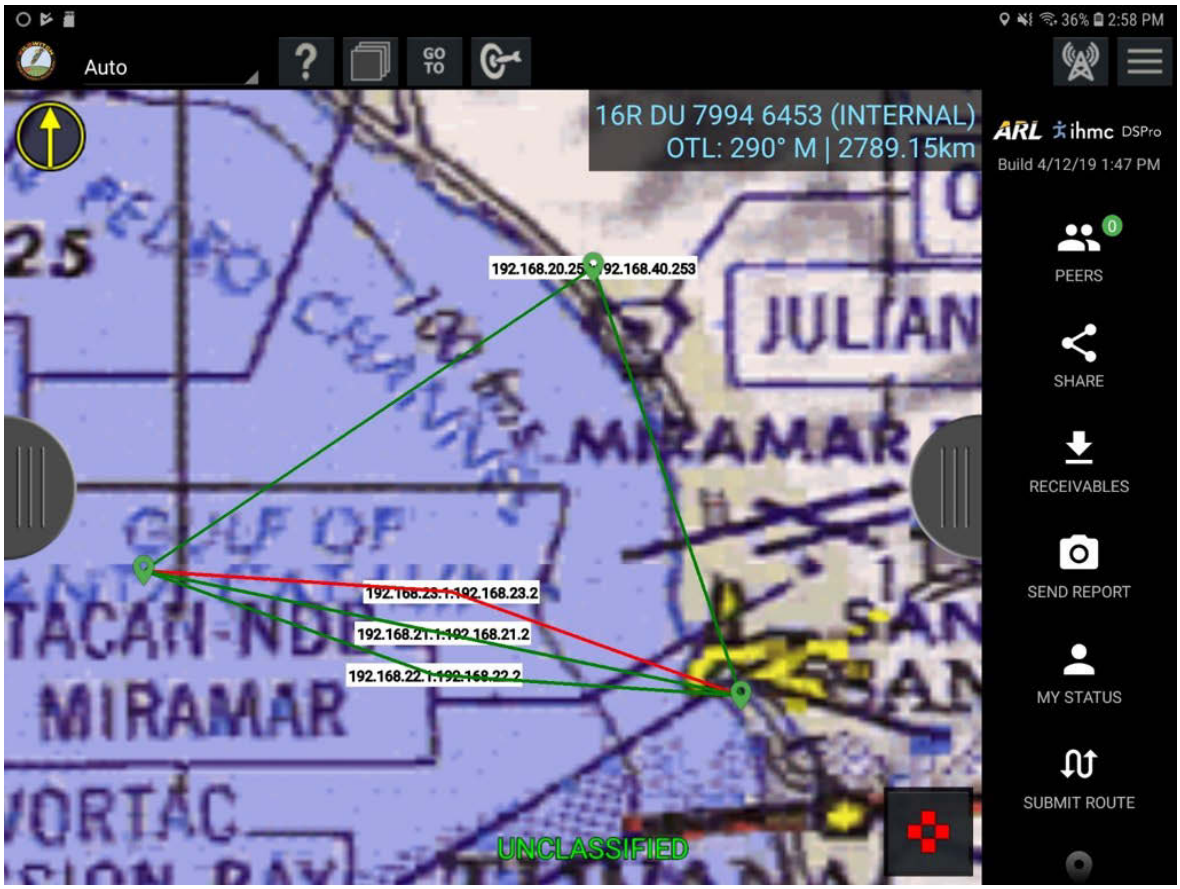



Figure 6.5: Integration of SENSEI network monitoring layer in Kilswitch.

### 6.1.7 OODA Loop

The Observe Orient Decide and Act (OODA) loop is a decision-making model first introduced by Boyd's and created by examining fighter pilots in aerial combat [113].

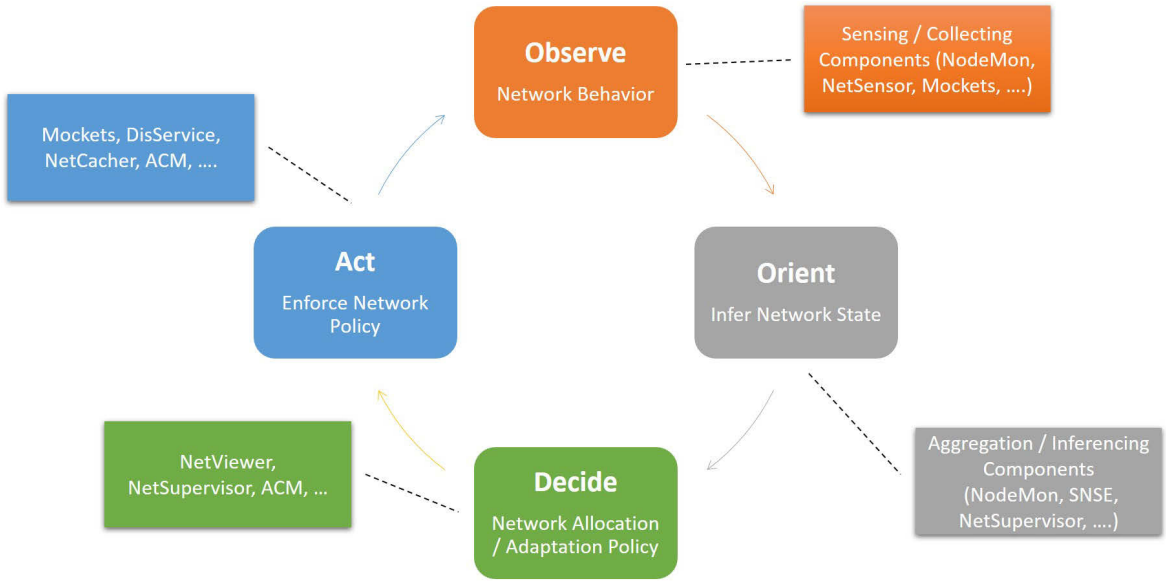



Figure 6.6: OODA Loop for Network Monitoring and Adaptation.

According to Boyd, the outside world is random and filled with ambiguities. Moreover, Boys argues that trying to understand a randomly changing space with pre-existing mental concepts can lead to confusion, ambiguity, and more uncertainty. From this, Boyd deduced that logical models of the surrounding reality are intrinsically incomplete, inconsistent, and must continuously be adapted according to newly obtained observations. In particular, Boyd encodes the process necessary to overcome this situation in a cyclical process made of four phases: Observe, Orient, Decide, and Act.

The Observe phase is used to harvest information about the environment of interest. This knowledge is then transformed, cross-referenced, and used to generate hypotheses on expected outcomes in the Orient phase. The most promising hypotheses are then used to formulate responses in the Decide phase. Finally, in the Act phase, the selected hypotheses and responses are tested by interacting with the environment.

It is worth noting that the phases are interconnected and guide each other in subsequent executions. For example, the Orient phase can guide the Observe phase to gain information to reinforce specific hypotheses. Finally, Boyd also emphasized the concept of Tempo, i.e., the time necessary to go through the four phases, arguing that operating at a faster Tempo than adversaries can prevent them from being effective by invalidating their assumption and creating confusion.

SENSEI implements what we call the OODA loop for network monitoring and adaptation. Figure 6.6 shows this loop. The figure shows that during the Observe phase, SENSEI harvests network information through specialized components (NetSensor) and from reports obtained sent by communication middleware (Mockets). During the Orient phase, NodeMonitor, SNSE, and NetSupervisor, aggregate, share, and process this information, to formulate hypotheses about the network status. In the Decide phase, NetSupervisor further processes the information creating actionable plans that will be executed during the Act phase. This cycle is continuously repeated since external events and the effect of the Act phase will change the network environments.

## 6.2 Status Exchange

### 6.2.1 World State

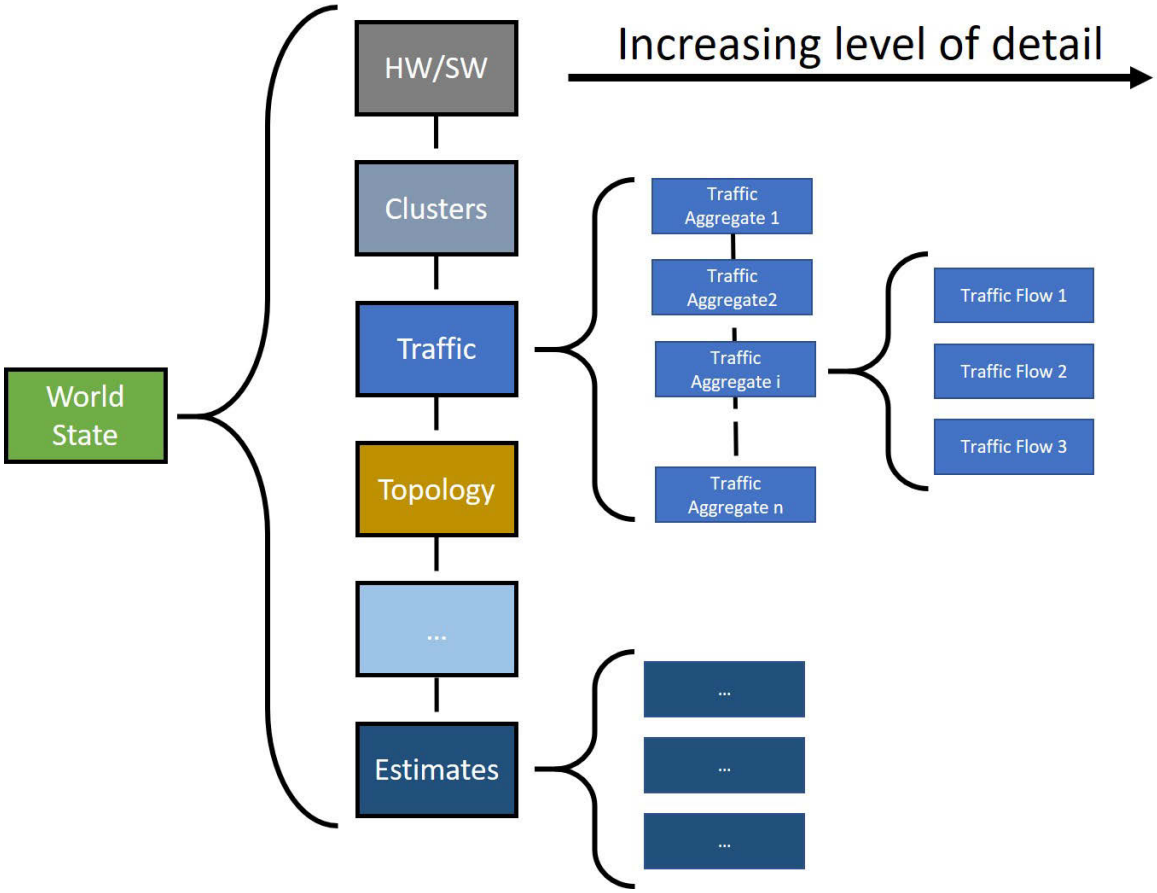



Figure 6.7: World State.

SENSEI stores information in the World State, an in-memory data structure that is independently maintained by each SNSE and that represents the state of the network as seen by each specific instance. The World State is composed of blocks each describing the consolidated view on a specific topic such as the status of a group of nodes, the traffic on some link, or the latency between two clusters. Each block is a hierarchical data structure built around a certain topic divided into sectors containing information at different levels of aggregation. For example, the most consolidated section on the traffic between two clusters only contains the name of the two clusters and the traffic between them, on the other hand, at a lower level of consolidation, all the reports used to create that consolidated view are maintained so that they can be requested if necessary. By default, only the sectors at the highest level of aggregation are periodically distributed between clusters to minimize network footprint.

Figure 6.7 shows a graphical representation of the World State. The picture shows several typical blocks that compose the World State, such as the “HW/SW” block that describes information about physical or close-to-the-machine characteristics of nodes belonging to the cluster, such as the installed hardware, e.g., CPU, RAM, network interfaces, the operating system, or the Clusters block that contains data about SENSEI instances, such as the addresses of detected sensors and known neighboring instances. The Traffic block shows multiple sectors at a different level of aggregation. In particular, the figure shows that ”Traffic Aggregate i” is made of

three Traffic flows that are aggregated into one to save information.

### 6.2.2 Data-agnostic aggregation

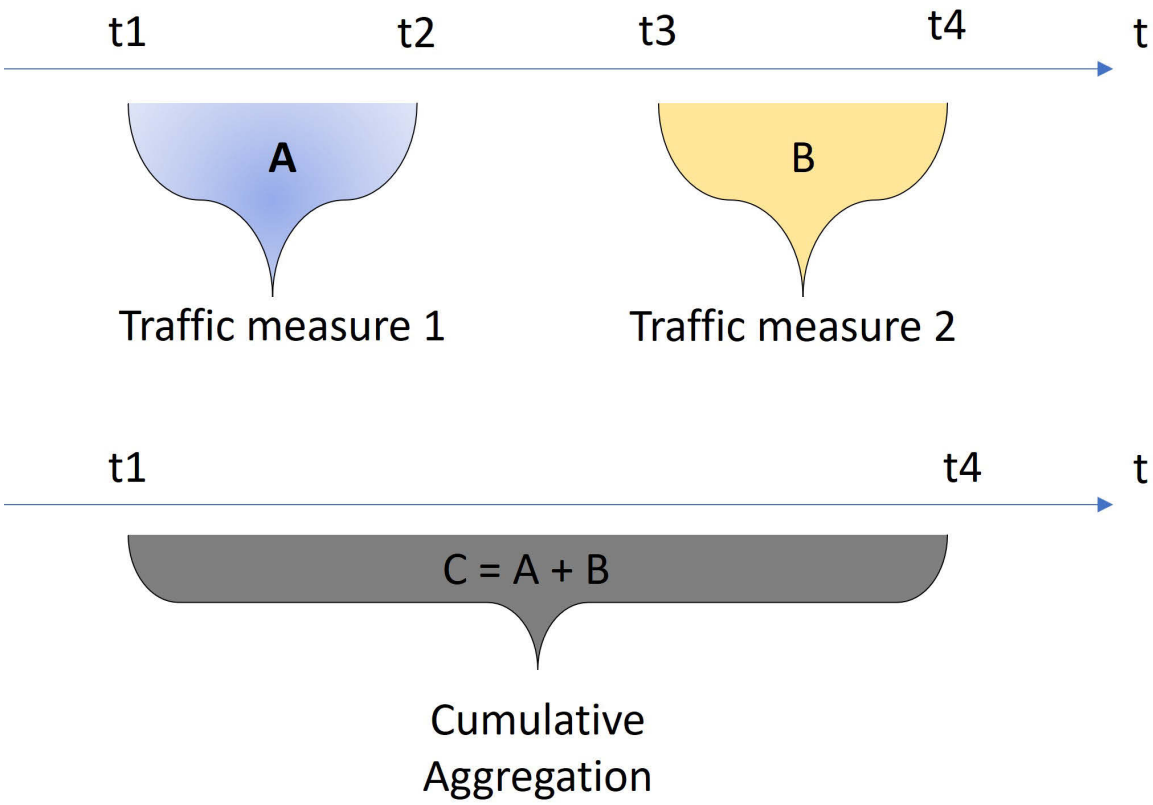



Figure 6.8: Example of cumulative aggregation.

Blocks are populated and replaced based on three types of data-agnostic aggregation policies that can be summarized as follows:

1. The first aggregation policy is used to aggregate non-cumulative information. Measures that arrive from a sensor within a certain time interval and are characterized by the same unique ID are added to a list.
2. The second aggregation policy is used for cumulative information. All the cumulative measures that fit within a certain time interval and are characterized by the same unique ID, are merged into a single measure. Figure 6.8 shows how two consecutive cumulative measures can be merged into a new measure by extending the time interval of the resulting measure and summing the values.
3. The third aggregation policy is used for cumulative information that cannot directly be aggregated by summing values such as merging the informative content of sensors reporting light and temperature levels in a room. In this case, both temperature and light are merged into a single measure, and their values are kept separated by using the sensor type to identify each value. This aggregation policy is used to remove the overhead associated with information coming from multiple low-level sensors describing the same ”object”.

Moreover, blocks are removed if no relevant information is received after a long time and replaced when new

information is received, i.e., newer than a time threshold, to give space to information that fits within the new time interval. These thresholds are automatically estimated by SNSE based on sensors' specific update patterns.

### 6.2.3 Content- and context- aware aggregation and distribution

SNSE implements several strategies to provide content- and context-aware aggregation and distribution with the help of NetSupervisor.

Redundant information is consolidated by merging entries that describe the same event observed by multiple sensors. In general, NetSupervisor resolves conflicts by prioritizing information obtained from sensors that are closer to the event under observation but it can also use custom algorithms for a subset of other information where that approach would not work.

Content awareness is achieved by prioritizing certain information over others on a per-destination basis. To do this, NetSupervisor uses knowledge about neighboring clusters, such as their associated subnetworks or hosts, to increase the chance that the shared information is relevant to each destination. For example, a traffic aggregate is by default only shared with neighboring clusters that are involved in the communication described by that aggregate.

Context-awareness can be achieved by modulating the quantity and type of shared information based on the communication link status. For instance, throughput thresholds can be calculated for each link based on the amount of bandwidth consumed by other applications that are sharing network resources.

SNSE then shares information accordingly with these limits by prioritizing highly-aggregated and critical information and only sharing lower priority data if there is sufficient extra bandwidth.

Moreover, SNSE incorporates a few different strategies to reduce the control overhead generated over the network. Updates are limited to 1 MTU-sized packet, and the update rate is adjusted dynamically based on the observed network capacity. Since the data available locally usually exceeds 1 MTU, subsets of the data are selected based on freshness and change rate. However, data that is skipped has its priority raised so that it will not be left out indefinitely.

### 6.2.4 World State Distribution

World State exchange between SENSEI instances is generally best-effort relying on multicast to share information between hosts belonging to the same network segment, and on Mockets to communicate with remote clusters that have to be reached through constrained and shared links.

We chose unicast for inter-cluster communication because network routers are often not configured to forward multicast traffic. Note that only one SNSE instance per cluster, called Cluster Master, can exchange data with remote nodes. We did this to increase control over the type and amount of traffic entering and leaving the cluster. Moreover, SNSE can be configured to act as a simple relay point to support uncommon configurations, and NetSupervisor can be configured to oversee multiple remote clusters.

The World State is exchanged between SENSEI instances using three policies: periodic, event-based, and request-based.

1. The **periodic** sharing policy is the default sharing mechanism used between Cluster Masters. Consistently with throughput and time limitations, each Cluster Master cycles the content of its World State sending only the most aggregate section of each block. A custom filtering function only allows relevant and aggregated information to be transmitted to neighbors. For example, if we have three Cluster Masters A, B, and C, and we indicate with $T_AB$ the traffic between A's and B's clusters, and with $T_AC$ the one between A's and C's, A will only send $T_AB$ to B and $T_AC$ to C. C will not be notified about $T_AB$, and B about $T_AC$.

2. **Event-based** sharing is used to distribute high-priority messages called Alerts. Currently, it is possible to configure static thresholds in SENSEI and assign them to specific metrics. During traffic monitoring, the measured values are compared to the configured threshold to catch anomalous situations and trigger information sharing with selected neighbors.

3. The **request-based** sharing policy enables remote Cluster Masters to subscribe and request specific blocks. This enables ACM's components to be notified about specific changes in the network conditions that the periodic policy would not include in its updates. For example. SENSEI visualization components exploit this mechanism to receive topology and traffic data from all clusters independently from their location.

### 6.2.5 Analysis of Default State Exchange Algorithm

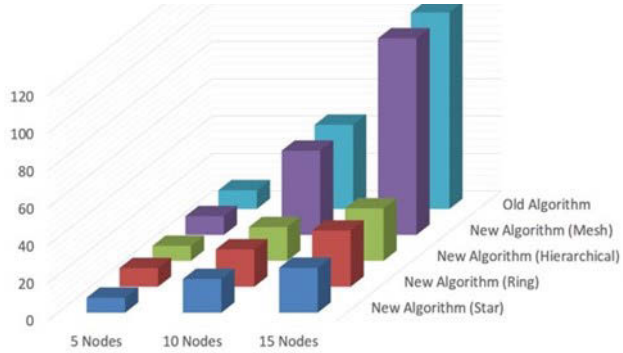



Figure 6.9: Exchanged CNS: comparison between old and new sharing algorithm with different network topologies.

In this simple analysis, we considered a network composed of N Clusters where each Cluster Master can compute a description of all the links with its neighbors. We also included a visualization component that makes requests to the local Cluster Master to receive and show the status of the whole network. Our first take on World State exchange had each Cluster Masters exchange information with any other. That algorithm could quickly saturate very constrained networks, and consequently, we developed an improved version that shares blocks only between Cluster Masters that are one-hop away from each other. It is worth noting that the network topology has a significant impact on the performance of the new algorithm.

In the case of mesh networks (where each node is directly connected with everyone else), the performance of the new and old algorithms are equivalent, however, the new algorithm will reduce the overhead due to block sharing in all other network topologies.

One drawback of the new algorithm is that Cluster Masters will not receive World State reports from nodes more than one-hop away. Nonetheless, in our experience, we noticed that it is unusual that nodes in a network need complete information concerning the status of distant nodes and that it is better to provide the information that is required through the request-based distribution policy which ensures that it is always possible to retrieve information relative to specific topics, regardless of the number of hops that separates two nodes.

Figure 6.9 shows a comparison between the number of reports periodically exchanged between Cluster Masters with the old and the new algorithm with varying network topologies. For example, for 15 nodes in a ring topology, the new algorithm outperforms the old one by sending less than 15% of the updates that the old algorithm would generate.

## 6.3 Passive Bandwidth Estimation

We designed SENSEI to passively detect several network metrics such as the traffic circulating the network, the RTT between hosts, and the available bandwidth. To provide passive bandwidth estimation, we designed Passive Packet Pair (P3), a passive version of the packet pair bandwidth estimation mechanism that assesses the bandwidth available over the end-to-end path by measuring differences in the inter-arrival times of packet pairs or trains produced at the transport protocol level starting from application-generated traffic. In particular, we integrated P3 into the Mockets communication middleware.

### 6.3.1 P3

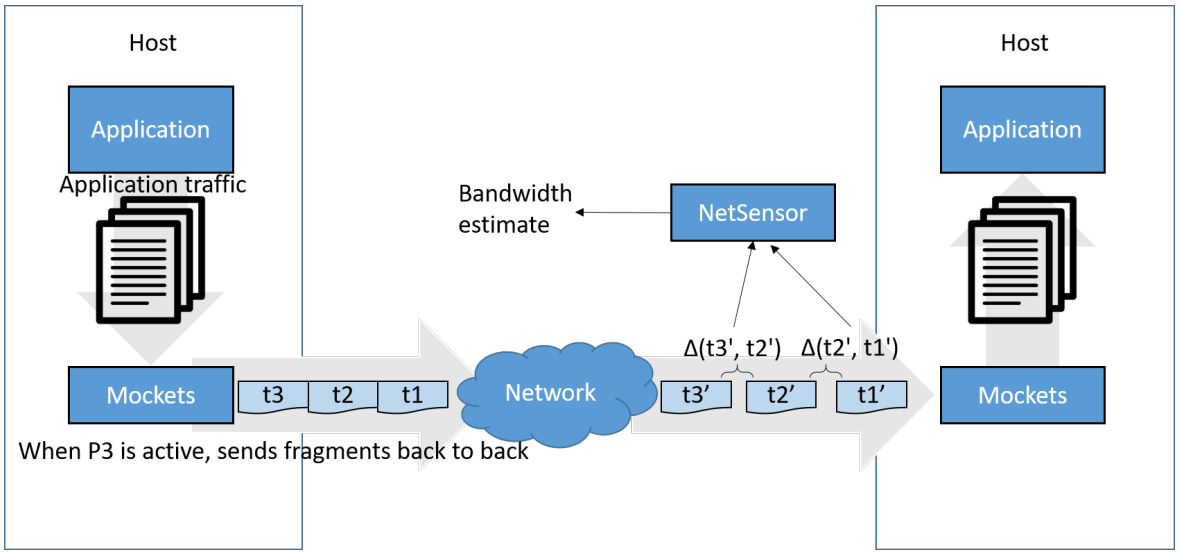



Figure 6.10: Bandwidth estimation using P3.

Similar to other approaches based on packet pairing, P3 estimates the available bandwidth over the bottleneck link between two endpoints by dividing the size (in bytes) of one of two consecutive packets by the time arrival difference of the last octet of each packet. For the estimates to be accurate, both packet pairs need to be of equal size and transmitted back to back by the sender.

P3 reduces statistical error by averaging time differences across multiple packet pairs, following an approach similar to Trains of Packet Pairs (TOPP) and SLoPS, but differently without requiring the introduction of prob-

ing traffic into the network. P3 can do this passive process because it exploits traffic exchanged by applications using Mockets.

Another significant difference that distinguishes P3 from other packet pair estimation techniques, is how the arrival time of the last octet of packets is measured. Traditional packet pair implementations assume that the application-level packet reception time, i.e., when the operating system delivers a packet to an application, coincides with the arrival time of the last octet of the packet. This approach suffers from a few issues and limitations:

1. The receiving application's behavior and the operating system's thread scheduling might negatively affect the measured times by introducing delays;
2. The receiving application is not aware of situations where packets from other flows are added between a pair's packets;
3. Measurements can only be taken on the node that runs the receiving application.

To mitigate these problems, P3 relies on NetSensor to measure packets' arrival time. Since NetSensor uses libpcap, it can acquire high fidelity packet arrival times based on the interface-generated packet timestamps or otherwise fallback to kernel-generated ones when not supported by the network interface.

Figure 6.10 illustrates the bandwidth estimation process performed by P3. In the figure, an application (on the left) using Mockets transfers some data, e.g., a document, to another host (on the right). The document is fragmented by Mockets into multiple packets that fit the network MTU and that are made ready for transmission. If the bandwidth needs to be estimated, Mockets marks all the headers of packets involved in the process with a special sequence of bytes that indicates that they are part of a pair or train for bandwidth estimation and the number of packets that will be part of the estimation. After that, Mockets pre-fetches all marked packets from memory to minimize the delay between transmissions and transmits them one after the other. By integrating P3 in Mockets, P3 can exploit the traffic exchanged by applications via the Mockets middleware instead of introducing probing packets to estimate the available bandwidth. While traversing the network, the packets originally transmitted back to back will accumulate a delay that depends on the capacity of the bottleneck link of the traveled path. This delay will take the form of inter-arrival time between packet pairs collected by NetSensor.

Upon receiving the first marked packet, the receiver enters a bandwidth estimation phase during which Mockets will block packet transmission for some time whose duration depends on the currently estimated bandwidth and the number of packets involved in the estimation. This allows P3 to reduce the chance of locally generated cross-traffic, e.g., due to message acknowledgments or other control or application traffic exchanged between the endpoints. This is important to increase the efficacy and the accuracy of P3 by maintaining a high ratio of valid samples, as P3 estimation calculations discard all measurements obtained from pairs that are afflicted by cross-traffic.

In particular, NetSensor records all packets flowing over the monitored links and stores each packet arrival time. Upon detecting a marked packet, NetSensor enters a bandwidth estimation session for the two endpoints and looks for other marked packets that belong to the same estimation sequence. If NetSensor detects cross-traffic

or packet reordering between probing packets, it drops the afflicted pair or train, and the sequence analysis continues from the next pair/train, if available. NetSensor can easily detect reordering, duplicates, and loss packets because each Mocket message is identified by an ever-increasing unique identifier.

Once the expected number of packets arrives, NetSensor computes the bandwidth estimation from the packet inter-arrival times and generates a report. After some time, if not enough packets are received in a certain interval, NetSensor aborts the estimation session and generates a report if at least one pair was received correctly.

P3 can either be done periodically or be requested by applications using Mockets. Additionally, Mockets can be configured to be completely passive or opportunistically passive. In the first case, Mockets will wait for a configurable interval of time for a sufficient amount of data to be made available for sending. If enough data is made available by the application before the specified time, Mockets will divide it into sequences (pairs or trains) of packets of equal size and send each sequence back to back. Otherwise, Mockets will abort the process. In the opportunistically passive case, if not enough traffic is available after the specified amount of time, Mockets proceeds by randomly generating enough data to reach the number of bytes required for the estimation.

## 6.4 Notes on SENSEI Deployment

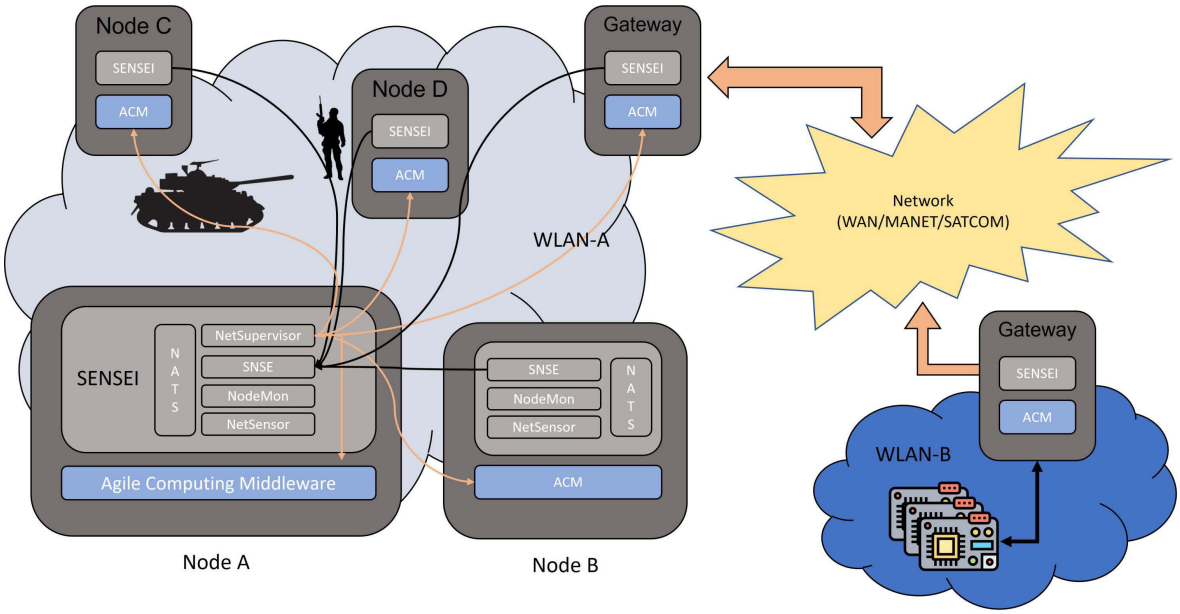



Figure 6.11: Example of deployment of SENSEI components over the nodes of a tactical network.

Figure 6.11 shows a possible deployment configuration for SENSEI. The picture shows a hybrid configuration where SENSEI instances are deployed in every node. In the picture, Node A is the Cluster Master of WLAN-A, in the figure, the node receives information from all the other SENSEI instances. Once processed, the information generated by SENSEI is distributed to the various communication middleware of the ACM to improve network utilization.

In general, we deploy SENSEI using one or more the following approaches depending on logistic and security constraints:

1. Full deployment;

2. Selective deployment;

3. Rely on network hardware.

### 6.4.1 Full deployment

In this configuration, SENSEI services are deployed in every host. This is the most pervasive solution, and it gives the most visibility and creates the most problems.

When sensors are deployed in every host, SENSEI can obtain a comprehensive picture of the system from all existing points of view. However, we can also expect redundancy caused by sensors observing the same events from different nodes within a cluster complicating the aggregation process and may not be applicable in particularly constrained sub-networks. For example, consider two sensors that monitor multicast traffic, one at the source and one at the destination.

Depending on the network technology and topology, aggregation algorithms have to recognize multicast packets and handle them correctly, avoiding traffic observed by multiple sensors to be counted more than once. These algorithms must also handle corner cases. For example. multicast traffic is sent as broadcast over Wi-Fi, while between switch hops it has to be counted multiple times.

To overcome latency between sensors and aggregation points, a timestamp is added to each report, and SENSEI uses this knowledge to time-align reports from different sensors. This requires that all the sensors generating reports have their time consistently aligned. Lack of synchronization could result in an incorrect interpretation of the results.

Security can also severely hinder a full-scale deployment of this kind. For example, some devices may not allow the use of root credentials [114] necessary to monitor network traffic using libpcap, in Android, we circumvented this limitation by creating an alternative version of NetSensor based on a custom VPN that would pass packets to NetSensor and mirror them in the device network interface instead of using libpcap. This solution has several disadvantages such as restricting other applications from being able to use VPN services and not being portable since it requires the use of a specific Android API.

### 6.4.2 Selective deployment

Sensors can also be placed in specific nodes. This has clear advantages in terms of configuration and deployment times. On the other hand, this solution reduces system visibility and suffers from some nuances in uncommon configurations. For example, SENSEI relies on the traffic shared by nodes to passively reconstruct the network topology, it follows that SENSEI can only build a topology of nodes for which it sees traffic. One compromise could be that of having sensors in choke points of the network and in selected internal nodes characterized by consistent interaction with other local nodes but that does not solve the problem of detecting nodes that are not generating traffic.

### 6.4.3 Rely on network hardware

Network observability can be provided by integrating NetSensor into local gateways. This approach is interesting when monitoring resource-constrained devices or when security prevents sensors deployment.

Moreover, network switches can be configured to mirror traffic to specific interfaces. NetSensor can then attach to these interfaces and monitor traffic from centralized locations. Note that with mirrored traffic, SENSEI cannot generally rely on sensor locations in the network to establish sensors' authority over network regions. One way to circumvent this problem is to override NetSensor defaults with values that are congruent with the network under observation. In general, knowledge has to be divided into meaningful sectors to maintain the content- and context-aware aggregation and distribution.

## 6.5 ACM

The ACM [11] is a suite of tools designed to provide efficient network communications, information dissemination, and opportunistic resource discovery in extremely challenging networking scenarios. In particular, this section presents NetCacher, a part of the ACM that once integrated with SENSEI can provide adaptive video streaming.

We developed NetCacher because traditional video streaming applications require large amounts of network resources and stable connections. Even considering modern advancements in video encoding and compression, such as High-Efficiency Video Coding [115], hundreds of Kbps are still necessary to stream videos of sufficient quality [48]. These traffic flows can compete with high priority and time-critical data generated by other applications sharing the same network resources. Nevertheless, TDR operations participants often desire video streaming as in some cases, it significantly enhances Situation Awareness. For example, a ground operator could use video surveillance provided by a UAV to spot or otherwise assess enemy activity or observe disaster survivors' status.

This and the fact that TDR operations are generally dynamic with varying requirements for information exchange calls for solutions capable of monitoring and interacting with communication middleware to balance congestion and increase the overall quality of the services provided. Competing usage also calls for systems that can dynamically change their bandwidth footprint and if possible, provide graceful degradation.

NetCacher is a video streaming management system specifically designed to overcome these challenges. Furthermore, this component is capable of real-time adaptation, providing parallel streaming at different qualities, and caching and forwarding video resources to overcome limitations related to node mobility and network disconnection.

### 6.5.1 NetCacher

Figure 6.12 summarizes NetCacher architecture which is composed of two elements, a streaming library that takes care of recording, transcoding, and distributing a video resource, and a streaming management environment that allows access and control of video resources by managing discovery, caching, and forwarding between NetCacher instances.

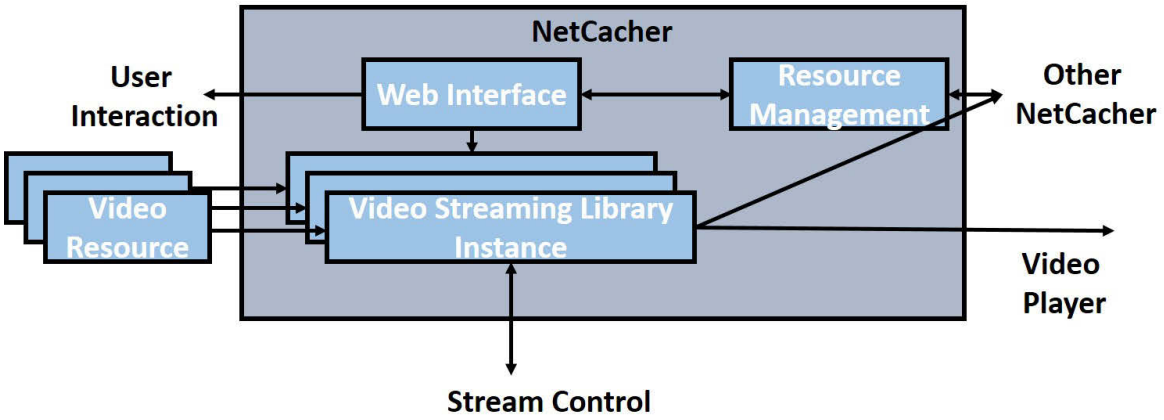



Figure 6.12: NetCacher architecture.

Each video resource is managed by one instance of the streaming library that records, transcodes, and sends the video to a remote receiver or another NetCacher allowing for the multiplexing of a single resource into parallel streams of different quality.

By processing SENSEI network monitoring information, NetCacher is capable of performing graceful degradation by balancing the trade-off between video quality and bandwidth consumption in real-time on a per-stream basis.

### 6.5.2 Streaming Library

NetCacher's Streaming Library is written in JAVA and leverages the OpenCV library to achieve a flexible video streaming service designed for the tactical environment. OpenCV is an open-source computer vision library developed by Intel [116] that facilitates the interaction with individual video streams' frames and wraps the FFmpeg library. NetCacher uses OpenCV to handle video encoding, decoding, streaming, and controlling video streams. Using OpenCV, the Streaming Library can interface with network video sources and consumers, such as IP cameras or WEB-based players, and become a source of its own (e.g. distributing multimedia contained in a file server).

The Streaming Library is composed of four logical parts, the Resource Recorder, the Resource Streamer, the Stream Controller, and the Stream Notifier, the Resource Recorder records and transcodes each resource into a stream and passes it to the Resource Streamer for transmission. Upon receiving a request for the resource, the Streaming Management Environment communicates to the Resource Streamer a unicast or multicast destination address for the stream.

For each active stream, the Stream Notifier advertises stream statistics to SENSEI. SENSEI can then communicate with the Stream Controller to control the stream, for example, by specifying a bandwidth limit. This feedback is then converted by the Stream Controller into what we call a stream profile, a compatible set of options for the video transmissions compatible with the feedback requirements.

NetCacher controls the video quality by using the OpenCV library, and in particular, we use the Video Buffering Verifier option to specify the quality and footprint of the video by setting a bandwidth limit.

### 6.5.3 Streaming Management Environment

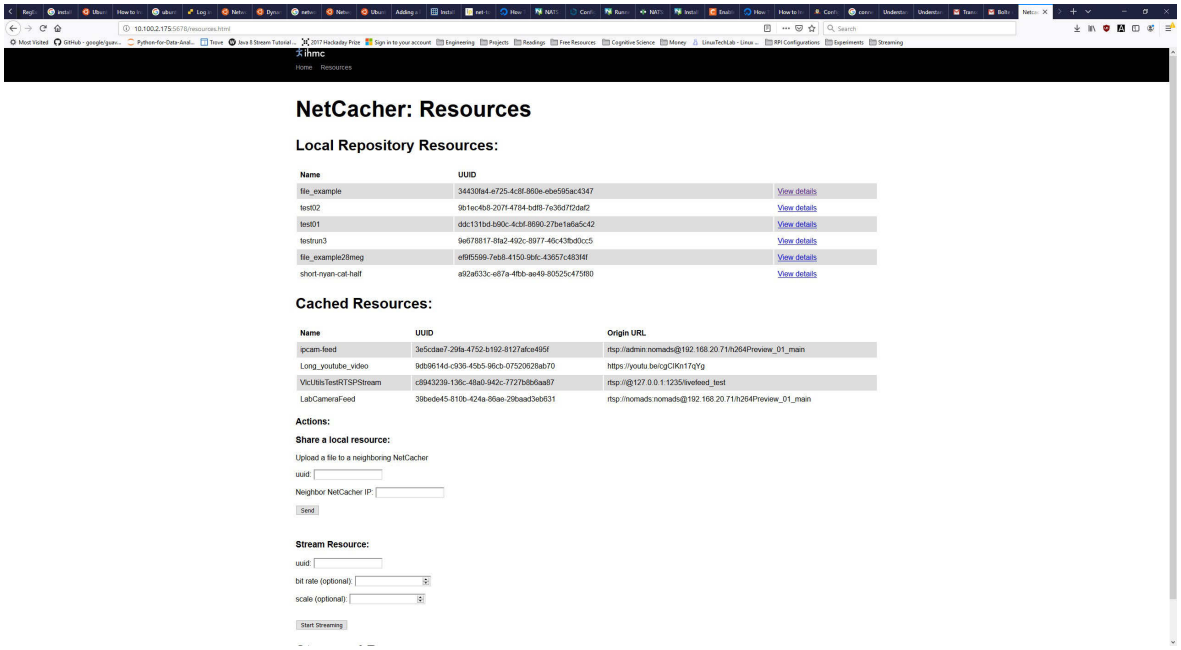



Figure 6.13: NetCacher WEB Interface.

The Streaming Management Environment provides a WEB interface (shown in Figure 6.13) to access, distribute, and remove video resources. Upon receiving a video streaming request, NetCacher creates a daisy-chained point-to-point re-streaming of the information that eventually reaches the requestor.

NetCacher can proactively distribute each stream to other instances to increase information liveness in environments characterized by nodes mobility. For example, a node can use this feature to cache video resources from a UAV to ground units when establishing a continuous link is not possible. Concurrent streams can be used to multiplex flows into streams of different quality, supporting, when necessary, networks of different capacities.

## 6.6 Experiments: Evaluation of Link Detection Algorithm

We designed the first set of experiments to test the reliability of NetSupervisor's link detection algorithm. The test was conducted using EMANE which we configured to emulate the characteristics of a degraded environment. More specifically we modeled three types of links, summarized in Table 6.2. In addition, we applied a 20% random noise to the statistics harvested by NetSensor, to test the stability of NetSupervisor's scoring process. The evaluation was conducted by randomly changing the characteristics of the link and the traffic exchanged between two subnetworks.

| Link ID | Bandwidth | Latency |
|---|---|---|
| WLAN | $< 100Mbps$ | $<= 10ms$ |
| SATCOM | $< 1Mbps$ | $>= 200ms$ |
| HF | $< 56Kbps$ | $<= 100ms$ |

Table 6.2: Links Exemplar Values.

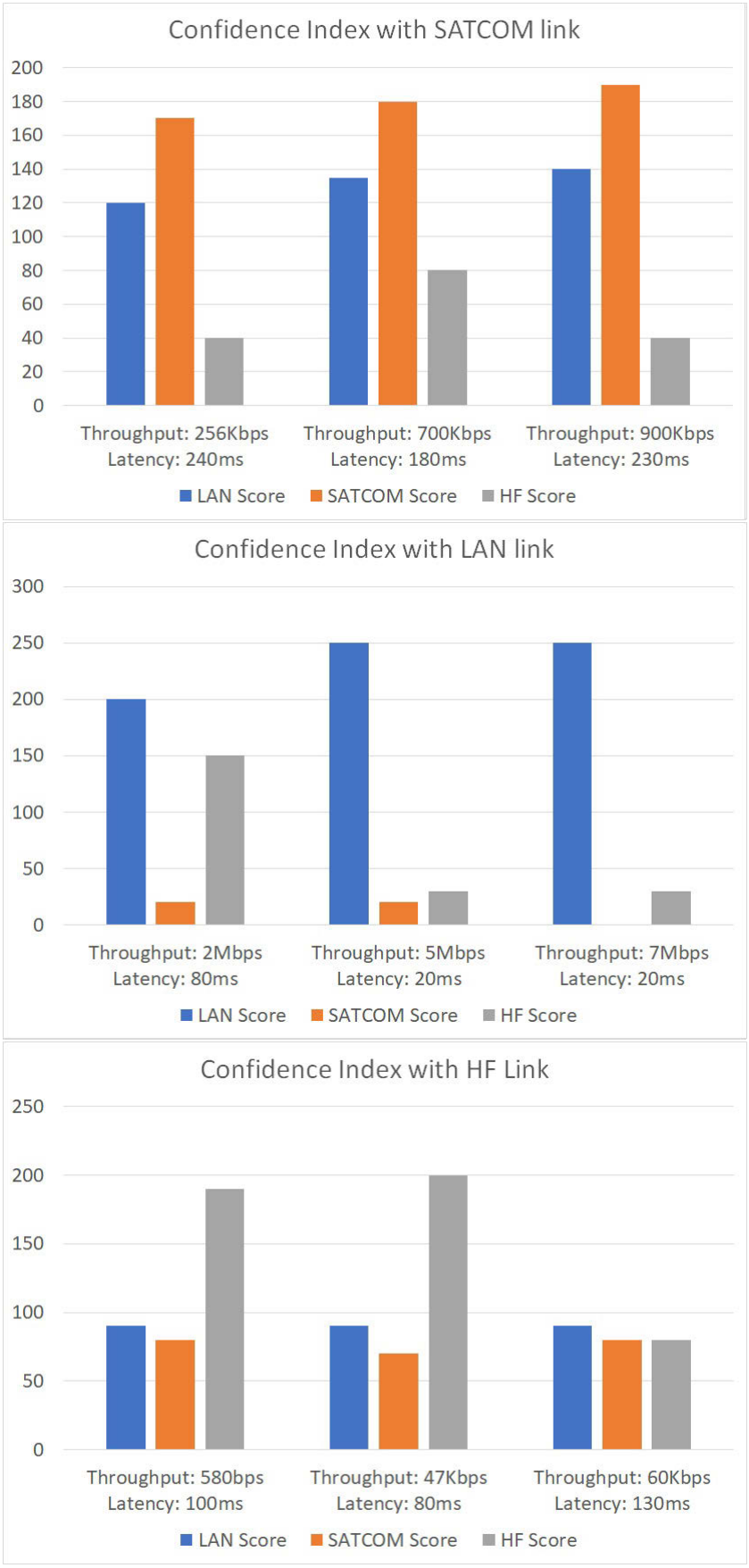



Figure 6.14: SATCOM (top), LAN (center), and HF (bottom) link score results.

Table 6.3 and 6.4 summarize the score tables used during the tests. Table 6.3 shows the score that would be

assigned to a specific link based on the value that was effectively detected. For example, the reference value for a LAN network is 100Mbps, for that link, if the observed throughput were to be larger than that value, the LAN link would get 100 points, if it were between 100 and 90Mbps, 90 points, and so on following the scores shown in the table. The same process applies to the other link types. Figure 6.14 shows the score results for different tests. NetSupervisor was correctly able to identify the link type 8 times out of 9 by a relevant margin which was measured as the difference in score with the other possible link types.

| Throughput Scores Rule | LAN | SATCOM | HF |
|---|---|---|---|
| $T >= ET$ | 100 | -20 | -20 |
| $0.9 * ET <= T < ET$ | 90 | 90 | 90 |
| $0.75 * ET <= T < ET$ | 90 | 100 | 100 |
| $0.50 * ET <= T < ET$ | 80 | 80 | 100 |
| $0.25 * ET <= T < ET$ | 70 | 70 | 100 |
| $0.1 * ET <= T < ET$ | 60 | 60 | 90 |
| $T < 0.1ET$ | 40 | 40 | 80 |

Table 6.3: Throughput scores used for the experiments.

| Latency Score Rules | LAN | SATCOM | HF |
|---|---|---|---|
| $0.8 * EL <= L < 1.2 * EL$ | 100 | 100 | 100 |
| $1.2 * EL <= L < 2 * EL$ | 100 | 100 | 100 |
| $L >= 2 * EL$ | 50 | 100 | 60 |
| $0.5 * EL <= L < 0.8 * EL$ | 90 | 40 | 70 |
| $0.2 * EL <= L < 0.5 * EL$ | 90 | 30 | 40 |
| $0.1 * EL <= L < 0.2 * EL$ | 90 | 20 | 30 |
| $L < 0.1 * EL$ | 90 | 10 | 20 |

Table 6.4: Latency scores used for the experiments.

NetSupervisor did not identify an outperforming HF link in the third experiment due to the added noise. Still, the fact that the inferred scores were very close, reveals the anomalous condition. In all the other experiments, NetSupervisor identified the link types correctly.

## 6.7 Experiments: Latency Detection Responsiveness

This set of experiments was conducted to evaluate SENSEI's latency detection responsiveness under different traffic loads. The results show that SENSEI can follow network variations consistently and that the overhead generated does not increase linearly with the number of nodes. In particular, we conducted two experiments. The first one measured the average latency of the Alert-based sharing policy with varying traffic loads. The second experiment measured the average bandwidth consumption of SENSEI while varying the number of nodes within the network. In the first experiment, we simulated 3 traffic loads using iPerf3 [117]: low (2560 bps), medium (128 Kbps), and high (204.8 Kbps). Using EMANE, we set up three clusters named TOC, LAV-1, and LAV-2, and controlled the links between the LAVs and the TOC. We then limited the link to 256 kbps and we randomly generated CE events modifying the latency between LAV-1 and TOC between 50ms and 1s

with a minimum change-step of 50 ms. We deployed SENSEI in the TOC and LAV-1 and extracted latency information using NetSensor's ICMP and TCP-RTT detection mechanisms.

Figure 6.15 shows that a small delay affects SENSEI's reports upon each event, but the error is otherwise small. The figure shows as a continuous line the real latency of the network and uses different shaped points to illustrate the results obtained by different RTT detection algorithms. Table 6.5, 6.6, and 6.7, summarize the errors and standard deviations with and without the "assessment tails".

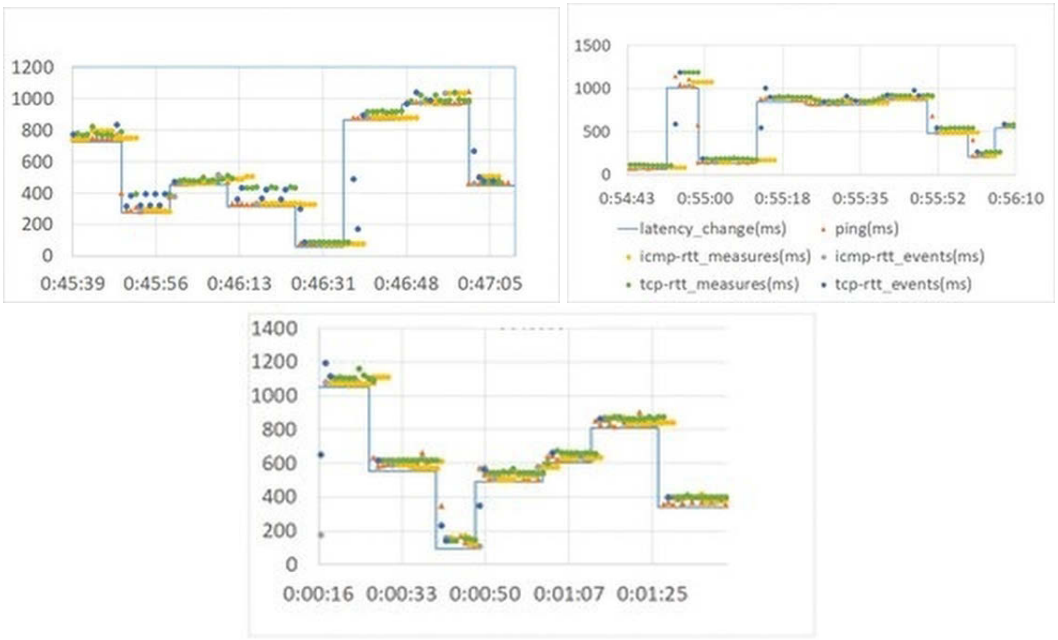



Figure 6.15: Latency comparison detected by SENSEI in three different configurations: 25600 bps (left), 128 kbps (right), and 205.8 kbps (bottom).

| Type | AVG Error | STD DEV | TAIL DELAY |
|---|---|---|---|
| ICMP | 124.4 ms | 235.8 | - |
| ICMP NT | 29 ms | 20.38 | 4s |
| TCP | 100.7 ms | 178.3 | - |
| TCP NT | 69.7 ms | 104.7 | 2 s |

Table 6.5: Error and Standard Deviation for the RTT detection mechanism with 25.6 Kbps. NT stands for No Tail.

| Type | AVG Error | STD DEV | TAIL DELAY |
|---|---|---|---|
| ICMP | 178.6 ms | 346.0 | - |
| ICMP NT | 32.8 ms | 15.97 | 4s |
| TCP | 118.0 ms | 215.6 | - |
| TCP NT | 60.5 ms | 32.6 | 2 s |

Table 6.6: Error and Standard Deviation for the RTT detection mechanism with 128 Kbps. NT stands for No Tail.

In the second experiment, we tested SENSEI in the Anglova scenario. In particular, we evaluated SENSEI overhead with 4, 8, 16, and 24 nodes, by measuring the average amount of traffic shared between nodes and

| Type | AVG Error | STD DEV | TAIL DELAY |
|---|---|---|---|
| ICMP | 116.47 ms | 192.3 | - |
| ICMP NT | 38,2 ms | 19.5 | 3s |
| TCP | 97.8 ms | 72.9 | - |
| TCP NT | 60.96 ms | 61 | 2 s |

Table 6.7: Error and Standard Deviation for the RTT detection mechanism with 294.8 Kbps. NT stands for No Tail.

companies. We evaluated the period-based sharing policy between 2 Cluster Masters where one of the clusters has a varying number of nodes within it.

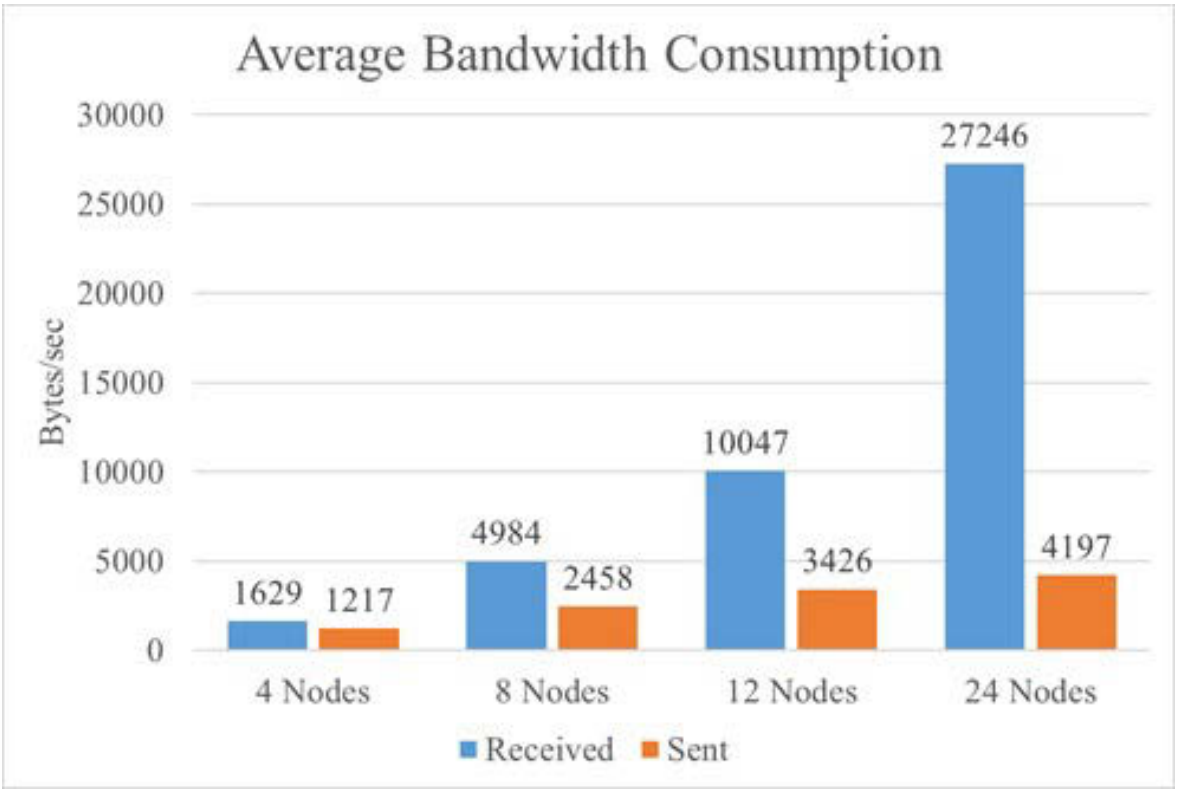



Figure 6.16: Bandwidth comparison inside one (blue) and between two companies (orange).

Figure 6.16 shows that there is a significant advantage in aggregating information as the number of nodes grows. The high reduction of expended bandwidth can be explained by considering that the three biggest contributors to the overhead in the company are topology and traffic information followed by a small amount of traffic used to keep the members in touch with each other. When the topology is fairly stable, SENSEI reverts to a state where it shares information every 10 seconds instead of every second removing any duplicated information. For example, if 2 nodes were to report the same topology, their views would be aggregated. Moreover, the compression is much more efficient at the Cluster Master level since it has more information to compress. Table 6.8 summarized these results.

| Nodes | Intra-Company Traffic (bytes) | Inter-Company Traffic (bytes) | Gain (%) |
|---|---|---|---|
| 4 | 1629 | 1217 | 25.3 |
| 8 | 4984 | 2458 | 50.6 |
| 12 | 10047 | 3426 | 65.9 |
| 24 | 27246 | 4197 | 84 |

Table 6.8: Summary of scalability with increasing number of SENSEI instances.

## 6.8 Experiments: Adaptive video streaming

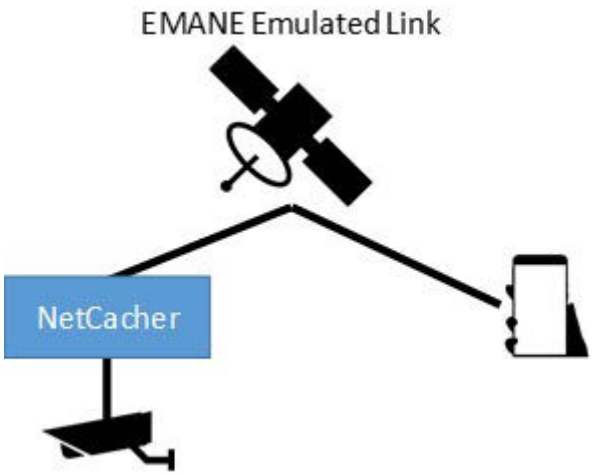



Figure 6.17: Adaptive Streaming experiment: A video multimedia is transmitted to a mobile device using a link with variable available bandwidth ranging from 128 Kbps to 1 Mbps.

We designed the fourth set of experiments to evaluate the effect of bandwidth on network stability managing a dynamic video stream through NetCacher and SENSEI. The two components collaborated to change video quality based on network conditions. Figure 6.17 shows the experiment topology. In the test, NetCacher would retrieve a video from an IP video camera, transcode it and stream it to a mobile device. We assumed an unconstrained link between NetCacher and the camera and we emulated the link between NetCacher and the mobile device using EMANE. During the experiment, we periodically changed the link characteristics so that the available bandwidth would go from 128 Kbps to 1 Mbps and vice-versa. A supporting application would generate TCP traffic through the same link (roughly 300 Bps, simulating blue-force data) and would measure delivery latency from one side of the link to the other.

For the first experiment, we manually controlled NetCacher to lower or increase video quality just before lowering or raising available bandwidth. The first plot of Figure 6.18 shows that by performing this manual adaptation the link never saturates and there is no increase in latency. In the second experiment, the video quality is only updated after SENSEI notices that the link is saturated. The second plot shows that the latency periodically and consistently increases to very high values signaling a saturation of the communication link. Seconds after, upon detecting the link saturation SENSEI suggests to NetCacher to lower its throughput. After some time the latency slowly goes back to normal preserving both the ability of the video to be streamed and restoring the timely delivery of traffic generated by the supporting application.

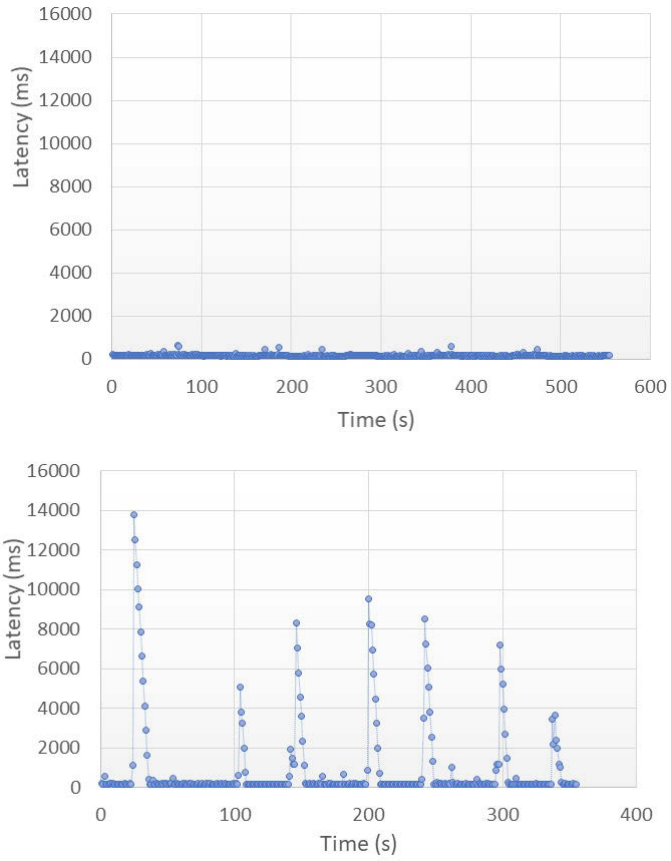



Figure 6.18: Latency spikes with manual (top) and dynamic (bottom) adaptation.

Figure 6.19 shows in bright green NetCacher traffic and in dark green the supporting application traffic. The figure shows that NetCacher shares 64 Kbps for the low-quality stream and 400 Kbps for the high-quality stream. The supporting application consistently shares 300 Bps. Figure 6.20 shows the difference in quality between the high- and low-quality streams with the image from the low-quality stream more pixelated than the high-quality one.

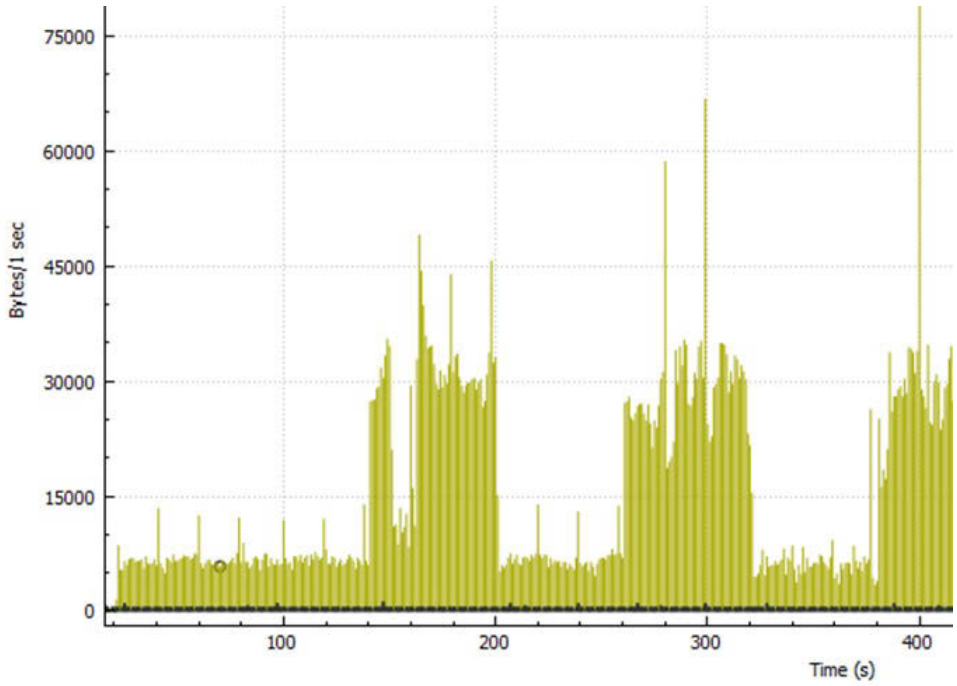



Figure 6.19: Traffic generated by NetCacher (Green) and by the support application (very small compared to the first). Increases in NetCacher traffic correspond to higher quality video streaming. Spikes are caused by limits of the FFmpeg library.

Figure 6.20: The left and right pictures show respectively a frame from a high and low video quality profile. As expected, pixelation is much more marked in the second frame.

## 6.9 Experiments: Evaluation of P3

This last set of experiments evaluates P3 over two separated experiments, one conducted in an emulated environment and one using physical radios. In both cases, two nodes exchange information using the Mockets communication middleware while NetSensor monitors packets to extract bandwidth estimation. This deployment is summarized in Figure 6.21. To drive the experimentation, we developed a simple client/server application that sends data from client to server using Mockets. On the server node, we also deployed NetSensor to monitor the packet trains generated by Mockets and extract their dispersion.

To simplify harvesting statistics, we developed a simple server capable of processing NetSensor results and generating reports. We emulated the first testbed using EMANE controlling the link between VMs utilizing CE. The emulated environment was composed of three Ubuntu VMs hosted in a VMWare ESXi hypervisor:

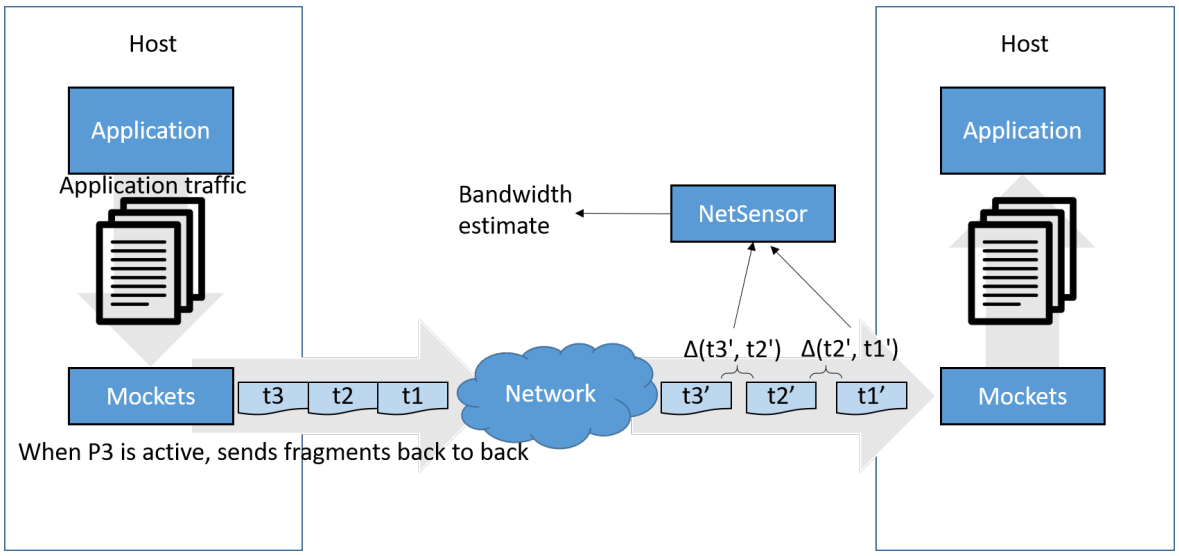



Figure 6.21: Experiment architecture.

two VMs were used to host the driver and one to host EMANE. Each VM had 2 CPUs and 2 GBs of RAM. We made the second testbed using two LEDE-based radios installed on a MikroTik RouterBoard RB411AH equipped with a Ubiquiti XR9 900 Mhz transceiver. Two wireless terminals were placed about 10 meters apart, separated by a building wall.

With our experiments, we wanted to verify four things. Firstly, we wanted to know if there was a correlation between CPU usage in our emulated environment and dispersion in the packet pairs. Knowing if there was correlation was important to decide if it was necessary to set hard resource limits in real deployments to avoid wrong measurements. Secondly, we wanted to verify that there was no significant penalty when using P3 in Mockets in terms of maximum throughput. Thirdly, we wanted to investigate what was the maximum bandwidth limit that we could reliably observe with our implementation of P3. Finally, our last experiment was designed to test the accuracy of P3 with real hardware.

### 6.9.1 Estimation Error over CPU

Figure 6.22 shows that we did not find a correlation between CPU usage and dispersion of detection accuracy. For this experiment, we configured the driver to send 5600 bytes of data with four probing packets (three pairs) per operation, and we repeated the experiment 10 times for each one of two CPU load values: very low (5%) CPU usage and very high (100%) CPU usage. We produced a total of 60 samples. The target CPU usage levels were achieved using the Linux *stress* utility configured to overload the 2 CPUs available on the VM. We verified the CPU loads from the ESXi control panel. We set the capacity of the emulated link to 512 Kbps, a value that did not introduce significant measurement errors (see Section 6.9.3 for more details on this). The figure shows the percentage of relative measurement error where the value of the real bandwidth is the one configured via EMANE and the measured value is the one obtained from P3 estimation. Each point is a sample harvested after re-running the experiment. The figure shows no apparent difference in the distribution of the error over the samples in the two configurations.

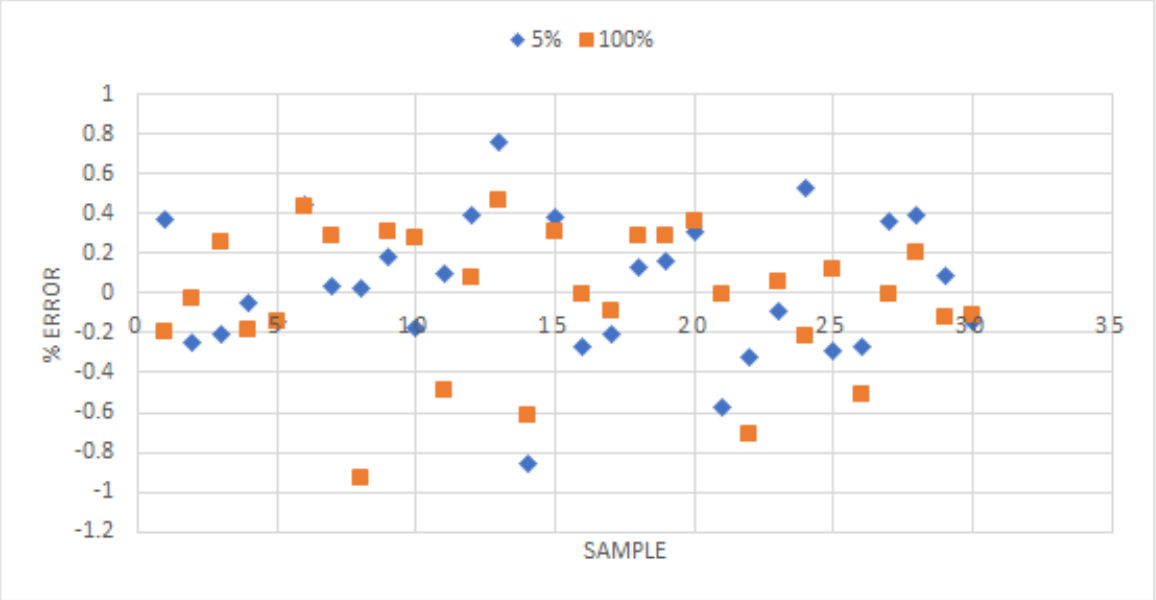



Figure 6.22: Samples' relative measurement error for 5% and 100% CPU load.

### 6.9.2 Estimation Overhead

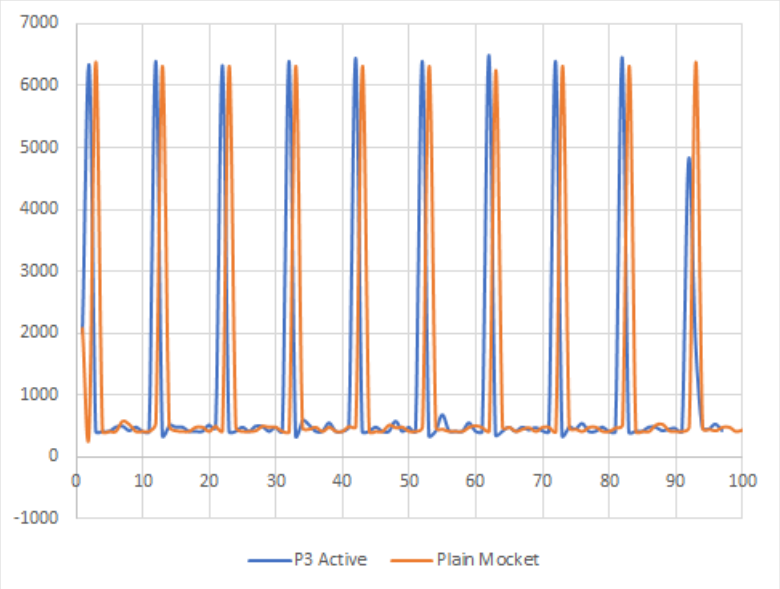



Figure 6.23: The plot shows bandwidth consumption with and without P3. The peak at the end is caused by a difference in the end of the experiments, what should be observed is the general trend.

To verify no significant penalty in using P3, we monitored the traffic exchanged between the server and the client using the tcpdump utility. We configured the driver to exchange 5600 bytes, and we repeated the experiment 10 times with P3 enabled and ten times with P3 disabled. Figure 6.23 shows that there is no significant difference in terms of generated traffic when comparing Mockets with and without P3. In practice, a few extra bytes are exchanged between the two endpoints to signal the start of the estimation process, but the packet trains are fully made of data that would have been exchanged anyway.

### 6.9.3 Estimation error over Bandwidth

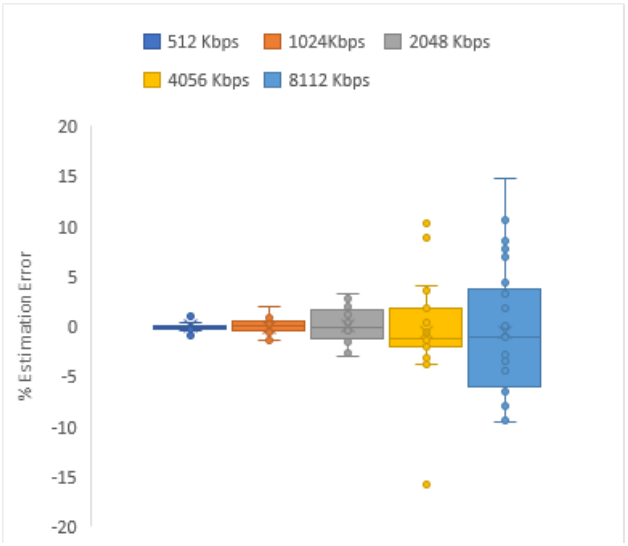



Figure 6.24: Error over bandwidth.

To measure the estimation limit of our P3 implementation, we exchanged 5600 bytes 10 times with different bandwidth settings in the emulated testbed and plotted the errors. We tried the following bandwidth values: 512, 1024, 2048, 4056, and 8112 Kbps; Figure 6.24 shows that our implementation remains fairly accurate until 2048 Kbps. After that value, statistical overshoot becomes more frequent and the error dispersion increases. One possible cause is how EMANE paces out packets injected into the network but this needs further investigation.

### 6.9.4 Estimation over radio Link

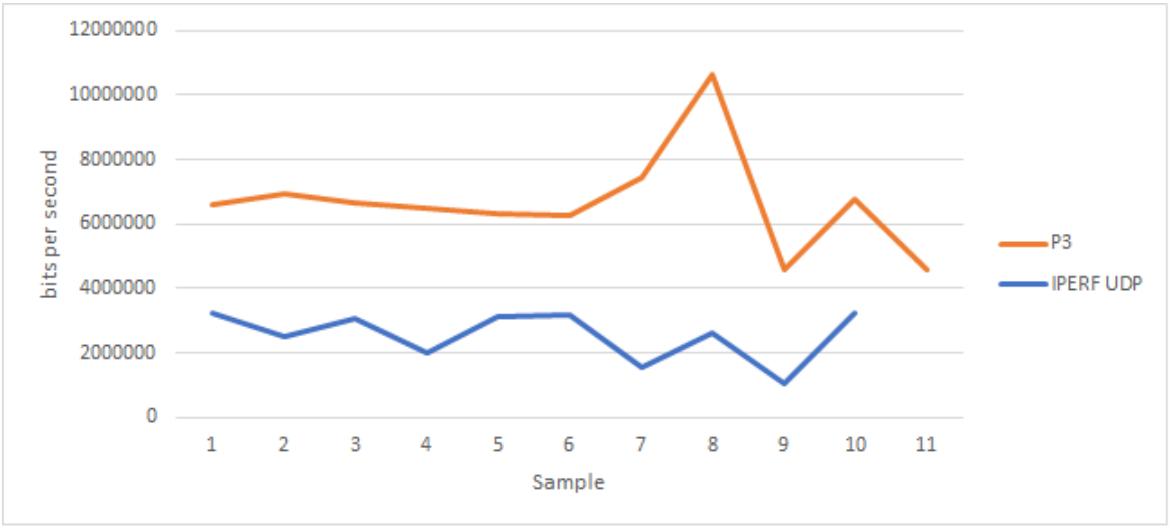



Figure 6.25: Radio Bandwidth Estimation between P3, and UDP iperf3.

To test P3 with real radios, we used our second testbed and we collected bandwidth estimation samples obtained when connecting our test endpoints using a radio link with a nominal throughput of 54000 kbps. We compared bandwidth estimates obtained using P3 and the Linux tool *iperf3*, configured to use UDP to transmit traffic at a rate of 8 MBps at the client-side and measure the received throughput at the server-side. Figure 6.25 shows

that P3 achieves greater accuracy than iperf3, which significantly underestimates the available capacity. The reason for such underestimate can be attributed to the high packet loss (about 40% or higher) experienced by iperf3, which inevitably lowers the measured throughput at the receiver's end. The reason for P3 overshoot is unclear and will require further investigation. One possibility is a queueing delay at the receiving side. That could reduce the delta between arrival times and explain the overshoot.

# Chapter 7

# Conclusion

This thesis presented results in three topics relevant to network-centric TDR operations conducted in degraded communication environments: unicast transport protocols, group communication solutions, and network status estimation and adaptation.

Chapter 4 was dedicated to research on unicast protocols and showed that compared with results obtained in past research [118], several valid COTS alternatives have emerged that can handle the high latencies, low bandwidths, and packet loss, that characterize TDR networks.

While these solutions may not provide the wide array of side capabilities that come with specialized libraries such as Mockets, the performance improvements are hard to argue against and a future can be envisioned in which improved versions of these COTS alternatives will reduce the need for custom protocols. Nonetheless, more experiments are required to confirm this. In particular, future experiments will need to test solutions with different bandwidth and latency variations in more dynamic scenarios because dynamic tests may better simulate the aleatory nature of tactical and disaster recovery missions.

For what concerns the research on group communication described in Chapter 5, there are several points to be made. Starting with emulation, this thesis presented several experiments using an 802.11ah and two SCB radio models designed to evaluate the performance of relevant COTS and custom-made group communication solutions. The SCB emulation enabled experiments in a realistic scenario capable of taking full advantage of multicast communication. In particular, this thesis presented two approaches. The first of the two is based on SMF and has the advantage of implementing a dynamically scheduled SCB which can efficiently handle traffic changes among the nodes. However, this method does not support the inclusion of cooperative transmission gains and only roughly model delay. Finally, given the reliance on SMF, which is a UDP-based protocol, it is very difficult to support applications that use TCP or other non-UDP protocols.

The second approach is based on precomputed topologies and needs to be pre-configured for the expected traffic loads. It requires precomputed topology data for each network size and parameter setting (which can require considerable preliminary work) but it may be preferable since it has the advantage of including cooperative transmission effects. Moreover, the delay estimates are much more accurate compared to the previous model. It is also worth noting that this model is not limited to UDP and can be used with any other transport protocol

supported by EMANE.

For what concerns GCS, this thesis presented an analysis conducted with the same objectives of the study on unicast protocols: to evaluate if COTS solutions could take over ones specifically designed for TDR operations. Each GCS was evaluated in terms of message delivery ratio, latency, and bandwidth utilization in a realistic environment supported by EMANE and the Anglova scenario using 802.11ah and PSCBT radio models.

The results obtained from these experiments indicate that ad hoc solutions specifically designed for constrained environments and based on multicast outperform all the other systems that we evaluated. More specifically, protocols designed for the internet did not provide adequate performance when deployed in a constrained scenario with characteristics similar to TDR operations which have low bandwidth, are unreliable, and exhibit variable latency.

The protocol analysis highlighted the existence of several significant differences in the design direction between consumer and tactical solutions. In particular, consumer applications tend to lean towards centralized TCP-based architectures, while tactical solutions towards custom implementations of reliable multicast. A possible explanation for this trend is that TCP is a well-known reliable protocol that works sufficiently well in performant networks. Moreover, centralized solutions are easier to develop and optimize. Still, multicast is intrinsically more efficient than unicast for group communication and can provide several advantages in constrained environments.

Considering the emerging prevalence of fog computing and IoT devices, there is no reason to argue against a future in which COTS solutions will be able to close the gap for group communication too. For this reason, the research described in this work should be repeated in the future when new solutions are available.

For what concern future research directions, there is a need to experiment with additional data types and with different delivery services, such as OLSRv2, to see if the added cost of running routing can compensate for scarce performance. Future experiments will also need to be executed with more nodes to evaluate scalability and the advantages of hierarchical dissemination structures. Additionally, more advanced forms of network degradation will need to be tested to evaluate if protocols can degrade graciously in the presence of bad networks. In particular, it could be interesting to compute results with variable jamming effects. More work also has to be done to better characterize the various trade-off that resulted from observing the existence of a trade-off between reliability and bandwidth consumption. In particular, bandwidth vs latency, bandwidth vs reliability, and reliability vs latency.

The third and last topic of this thesis discussed in Chapter 6 is that of monitoring and adaptation in TDR operations environments. In particular, this thesis argued that traffic monitoring is of great interest to identify anomalies and adapt to changing network conditions but since networks used to support TDR operations are severely degraded and bandwidth depleted, it is essential to conserve bandwidth by opportunely reducing the volume of monitoring traffic and by preferring passive detection approaches whenever possible. To this avail, this thesis presented SENSEI, a distributed, resource-frugal solution capable of passively monitoring the network, inferring its status, and driving the behavior of other components. SENSEI has the potential to greatly enhance the performance of applications deployed in degraded environments by enabling them to adapt their behavior in response to communication links' status changes, for example by automatically reducing the output

of low priority traffic in congested channels.

In Chapter 6, this thesis also illustrated how SENSEI passively harvests several network characteristics such as latency (relying on other communication middleware and traffic generated by other nodes), and bandwidth (through the Passive Packet Pair algorithm implemented in NetSensor and Mockets). Moreover, the same chapter presented NetCacher, a solution capable of dynamically transcoding video streams based on network usage reports obtained from SENSEI. Through this example, this thesis argued that many of the characteristics of the tactical scenario limit the use of mainstream software and discussed possible improvements showing that network monitoring needs to be more specialized and consider other participants' needs (not only the end-to-end quality of a video stream).

# Bibliography


[1] Florian Grandhomme et al. “ITMAN: An inter tactical mobile ad hoc network routing protocol”. In: *MILCOM 2016 - 2016 IEEE Military Communications Conference*. 2016, pp. 823–828. DOI: 10.1109/MILCOM.2016.7795431.

[2] Cardwell Neal et al. “BBR: Congestion-Based Congestion Control”. In: *ACM Queue*. Vol. 14. 5. 2016, pp. 20–53.

[3] Martin Thomson. *Version-Independent Properties of QUIC*. RFC 8999. May 2021. DOI: 10.17487/RFC8999. URL: https://rfc-editor.org/rfc/rfc8999.txt.

[4] Vignesh Sridharan et al. “Exploring Performance Trade-offs in Tactical Edge Networks”. In: *IEEE Communications Magazine* 58.8 (2020), pp. 28–33. DOI: 10.1109/MCOM.001.2000303.

[5] Niranjan Suri et al. “Communications middleware for tactical environments: Observations, experiences, and lessons learned”. In: *IEEE Communications Magazine* 47.10 (2009), pp. 56–63. DOI: 10.1109/MCOM.2009.5273809.

[6] Bomin Mao et al. “An Intelligent Packet Forwarding Approach for Disaster Recovery Networks”. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761638.

[7] Matthias Herlich and Shigeki Yamada. “Comparing Strategies to Construct Local Disaster Recovery Networks”. In: *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. 2016, pp. 376–383. DOI: 10.1109/AINA.2016.74.

[8] Niranjan Suri et al. “Peer-to-peer communications for tactical environments: Observations, requirements, and experiences”. In: *IEEE Communications Magazine* 48.10 (2010), pp. 60–69. DOI: 10.1109/MCOM.2010.5594678.

[9] Mihoko Sakurai et al. “Sustaining life during the early stages of disaster relief with a frugal information system: learning from the great east Japan earthquake”. In: *IEEE Communications Magazine* 52.1 (2014), pp. 176–185. DOI: 10.1109/MCOM.2014.6710081.

[10] M. Grötschel, C. Raack, and A. Werner. “Towards optimizing the deployment of optical access networks”. In: *EURO Journal on Computational Optimization* 2 (2014), pp. 17–53.

[11] Giacomo Benincasa et al. “Agile Communication Middleware for Next-Generation Mobile Heterogeneous Networks”. In: *IEEE Software* 31.2 (2014), pp. 54–61. DOI: 10.1109/MS.2013.144.

[12] Mauro Tortonesi et al. “Enabling the deployment of COTS applications in tactical edge networks”. In: *IEEE Communications Magazine* 51.10 (2013), pp. 66–73. DOI: 10.1109/MCOM.2013.6619567.

[13] Niranjan Suri et al. “Communications middleware for tactical environments: Observations, experiences, and lessons learned”. In: *IEEE Communications Magazine* 47.10 (2009), pp. 56–63. DOI: 10.1109/MCOM.2009.5273809.

[14] Kelvin Marcus et al. “Evaluation of the scalability of OLSRv2 in an emulated realistic military scenario”. In: *2017 International Conference on Military Communications and Information Systems (ICMCIS)*. 2017, pp. 1–8. DOI: 10.1109/ICMCIS.2017.7956503.

[15] Cesare Stefanelli et al. “Network Conditions Monitoring in the Mockets Communications Framework”. In: *MILCOM 2007 - IEEE Military Communications Conference*. 2007, pp. 1–7. DOI: 10.1109/MILCOM.2007.4454899.

[16] Richard Carl et al. “Transport Protocols in the Tactical Network Environment”. In: *2007 IEEE Aerospace Conference*. 2007, pp. 1–9. DOI: 10.1109/AERO.2007.352905.

[17] H L Gururaj and B Ramesh. “Effect of link level impairments on transmission control protocol in military networks”. In: *2016 IEEE Annual India Conference (INDICON)*. 2016, pp. 1–7. DOI: 10.1109/INDICON.2016.7838985.

[18] Yanping Teng et al. “A Study of Improved Approaches for TCP Congestion Control in Ad Hoc Networks”. In: *Procedia Engineering* 29 (2012). 2012 International Workshop on Information and Electronics Engineering, pp. 1270–1275. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2012.01.125. URL: https://www.sciencedirect.com/science/article/pii/S187770581200135X.

[19] Andreas R. Urke, Lars Erling Bråten, and Knut Øvsthus. “TCP challenges in hybrid military satellite networks; measurements and comparison”. In: *MILCOM 2012 - 2012 IEEE Military Communications Conference*. 2012, pp. 1–6. DOI: 10.1109/MILCOM.2012.6415561.

[20] M. Bell et al. “Explicit Congestion Control for Efficient Reliable Transport in IP-based Tactical Networks”. In: *MILCOM 2006 - 2006 IEEE Military Communications conference*. 2006, pp. 1–7. DOI: 10.1109/MILCOM.2006.302309.

[21] Ankita V. Terkhedkar and Medha A. Shah. “Approaches to provide security in publish/subscribe systems — A review”. In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 1. 2016, pp. 1–4.

[22] A. Scaglione and Yao-Win Hong. “Opportunistic large arrays: cooperative transmission in wireless multihop ad hoc networks to reach far distances”. In: *IEEE Transac-*

*tions on Signal Processing* 51.8 (2003), pp. 2082–2092. DOI: 10.1109/TSP.2003.814519.

[23] Adam Blair et al. "Barrage relay networks for cooperative transport in tactical MANETs". In: *MILCOM 2008 - 2008 IEEE Military Communications Conference*. 2008, pp. 1–7. DOI: 10.1109/MILCOM.2008.4753655.

[24] Jimmi Grönkvist et al. "Dynamic scheduling for cooperative broadcasting in tactical ad hoc networks". In: *MILCOM 2016 - 2016 IEEE Military Communications Conference*. 2016, pp. 1034–1040. DOI: 10.1109/MILCOM.2016.7795466.

[25] Arwid Komulainen, Jimmi Grönkvist, and Jan Nilsson. "Comparing the capacity of cooperative broadcast to spatial reuse TDMA in ad hoc networks". In: *2016 International Conference on Military Communications and Information Systems (ICMCIS)*. 2016, pp. 1–7. DOI: 10.1109/ICMCIS.2016.7496581.

[26] Ta Chen et al. "Enhancing application performance with network awareness in Tactical Networks". In: *2011 - MILCOM 2011 Military Communications Conference*. 2011, pp. 1158–1163. DOI: 10.1109/MILCOM.2011.6127456.

[27] Andy S. Peng et al. "Automatic Dynamic Resource Management architecture in tactical network environments". In: *MILCOM 2009 - 2009 IEEE Military Communications Conference*. 2009, pp. 1–7. DOI: 10.1109/MILCOM.2009.5379748.

[28] R. Prasad et al. "Bandwidth estimation: metrics, measurement techniques, and tools". In: *IEEE Network* 17.6 (2003), pp. 27–35. DOI: 10.1109/MNET.2003.1248658.

[29] J. D. Case et al. "RFC1157: Simple Network Management Protocol (SNMP)". In: USA: RFC Editor, 1990.

[30] Giri Kuthethoor et al. "Performance analysis of SNMP in airborne tactical networks". In: *MILCOM 2008 - 2008 IEEE Military Communications Conference*. 2008, pp. 1–7. DOI: 10.1109/MILCOM.2008.4753386.

[31] Giri Kuthethoor et al. "Performance analysis of SNMP in airborne tactical networks". In: Dec. 2008, pp. 1–7. DOI: 10.1109/MILCOM.2008.4753386.

[32] Liang Ma et al. "Inferring Link Metrics From End-To-End Path Measurements: Identifiability and Monitor Placement". In: *IEEE/ACM Transactions on Networking* 22.4 (2014), pp. 1351–1368. DOI: 10.1109/TNET.2014.2328668.

[33] Liang Ma et al. "Monitor placement for maximal identifiability in network tomography". In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 2014, pp. 1447–1455. DOI: 10.1109/INFOCOM.2014.6848079.

[34] *NetNORAD: Troubleshooting networks via end-to-end probing*. https://engineering.fb.com/2016/02/18/core-data/netnorad-troubleshooting-networks-via-end-to-end-probing/. Accessed: 09/18/2021.

[35] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. "Assolo, a New Method for Available Bandwidth Estimation". In: *2009 Fourth International Conference on Internet Monitoring and Protection*. 2009, pp. 130–136. DOI: 10.1109/ICIMP.2009.28.

[36] Taner Arsan. “Review of bandwidth estimation tools and application to bandwidth adaptive video streaming”. In: *High Capacity Optical Networks and Emerging/Enabling Technologies*. 2012, pp. 152–156. DOI: 10.1109/HONET.2012.6421453.

[37] Sukhpreet Kaur Khangura and Markus Fidler. “Available bandwidth estimation from passive TCP measurements using the probe gap model”. In: *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. 2017, pp. 1–9. DOI: 10.23919/IFIPNetworking.2017.8264826.

[38] Sukhpreet Kaur Khangura. “Neural Network-based Available Bandwidth Estimation from TCP Sender-side Measurements”. In: *2019 8th International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*. 2019, pp. 1–6. DOI: 10.23919/PEMWN47208.2019.8986907.

[39] Yufeng Xiao et al. “A New Available Bandwidth Measurement Method Based on Self-Loading Periodic Streams”. In: *2007 International Conference on Wireless Communications, Networking and Mobile Computing*. 2007, pp. 1904–1907. DOI: 10.1109/WICOM.2007.477.

[40] Abdelaziz Amamra and Kun Mean Hou. “Available Bandwidth Estimation in Wireless Ad Hoc Network: Accuracy and Probing Time”. In: *2008 11th IEEE International Conference on Computational Science and Engineering*. 2008, pp. 379–387. DOI: 10.1109/CSE.2008.57.

[41] Jacob Strauss, Dina Katabi, and Frans Kaashoek. “A Measurement Study of Available Bandwidth Estimation Tools”. In: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC ’03. Miami Beach, FL, USA: Association for Computing Machinery, 2003, pp. 39–44. ISBN: 1581137737. DOI: 10.1145/948205.948211. URL: https://doi.org/10.1145/948205.948211.

[42] Takashi Oshiba and Kazuaki Nakajima. “Quick end-to-end available bandwidth estimation for QoS of real-time multimedia communication”. In: *The IEEE symposium on Computers and Communications*. 2010, pp. 162–167. DOI: 10.1109/ISCC.2010.5546789.

[43] R. de Renesse et al. “QoS enabled routing in mobile ad hoc networks”. In: *Fifth IEE International Conference on 3G Mobile Communication Technologies*. 2004, pp. 678–682. DOI: 10.1049/cp:20040762.

[44] Yaling Yang and R. Kravets. “Contention-aware admission control for ad hoc networks”. In: *IEEE Transactions on Mobile Computing* 4.4 (2005), pp. 363–377. DOI: 10.1109/TMC.2005.52.

[45] Cheikh Sarr et al. “Bandwidth Estimation for IEEE 802.11-Based Ad Hoc Networks”. In: *IEEE Transactions on Mobile Computing* 7.10 (2008), pp. 1228–1241. DOI: 10.1109/TMC.2008.41.

[46] Nam Van Nguyen et al. “Retransmission-Based Available Bandwidth Estimation in IEEE 802.11-Based Multihop Wireless Networks”. In: *Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and*

*Mobile Systems*. MSWiM '11. Miami, Florida, USA: Association for Computing Machinery, 2011, pp. 377–384. ISBN: 9781450308984. DOI: 10.1145/2068897.2068961. URL: https://doi.org/10.1145/2068897.2068961.

[47] Redouane Belbachir, Zoulikha Mekkakia Maaza, and Ali Kies. "Towards a New Approach in Available Bandwidth Measures on Mobile Ad Hoc Networks". In: *Acta Polytechnica Hungarica* 8 (Oct. 2011), pp. 113–128.

[48] Margaret H. Pinson, Stephen Wolf, and Robert B. Stafford. "Video Performance Requirements for Tactical Video Applications". In: *2007 IEEE Conference on Technologies for Homeland Security*. 2007, pp. 85–90. DOI: 10.1109/THS.2007.370025.

[49] Scott Pudlewski et al. "Video Transmission Over Lossy Wireless Networks: A Cross-Layer Perspective". In: *IEEE Journal of Selected Topics in Signal Processing* 9.1 (2015), pp. 6–21. DOI: 10.1109/JSTSP.2014.2342202.

[50] James Nightingale et al. "Reliable full motion video services in disadvantaged tactical radio networks". In: *2016 International Conference on Military Communications and Information Systems (ICMCIS)*. 2016, pp. 1–9. DOI: 10.1109/ICMCIS.2016.7496560.

[51] Zhenghua Fu, Xiaoqiao Meng, and Songwu Lu. "How bad TCP can perform in mobile ad hoc networks". In: *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*. 2002, pp. 298–303. DOI: 10.1109/ISCC.2002.1021693.

[52] M. Rodríguez-Pérez et al. "Common problems in delay-based congestion control algorithms: a gallery of solutions". In: *European Transactions on Telecommunications* 22.4 (2011), pp. 168–178. DOI: https://doi.org/10.1002/ett.1485. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.1485. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.1485.

[53] Jianchuan Zhang et al. "A Survey of TCP Congestion Control Algorithm". In: *2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP)*. 2020, pp. 828–832. DOI: 10.1109/ICSIP49896.2020.9339423.

[54] Mikel Irazabal et al. "Dynamic Buffer Sizing and Pacing as Enablers of 5G Low-Latency Services". In: *IEEE Transactions on Mobile Computing* (2020), pp. 1–1. DOI: 10.1109/TMC.2020.3017011.

[55] Jae Chung and M. Claypool. "Analysis of active queue management". In: *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.* 2003, pp. 359–366. DOI: 10.1109/NCA.2003.1201176.

[56] Marco Happenhofer and Peter Reichl. "Measurement-Based Analysis of Head-of-Line Blocking for SIP over TCP". In: *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 2010, pp. 244–253. DOI: 10.1109/MASCOTS.2010.33.

[57] P.K. Chowdhury, M. Atiquzzaman, and W. Ivancic. “Handover schemes in space networks: classification and performance comparison”. In: *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*. 2006, 8 pp.–108. DOI: 10.1109/SMC-IT.2006.33.

[58] Michael Welzl and David Ros. *A Survey of Lower-than-Best-Effort Transport Protocols*. RFC 6297. June 2011. DOI: 10.17487/RFC6297. URL: https://rfc-editor.org/rfc/rfc6297.txt.

[59] Andrei Gurtov et al. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 6582. Apr. 2012. DOI: 10.17487/RFC6582. URL: https://rfc-editor.org/rfc/rfc6582.txt.

[60] K.N. Srijith, L. Jacob, and A.L. Ananda. “Worst-case performance limitation of TCP SACK and a feasible solution”. In: *The 8th International Conference on Communication Systems, 2002. ICCS 2002.* Vol. 2. 2002, 1152–1156 vol.2. DOI: 10.1109/ICCS.2002.1183313.

[61] M. Gerla et al. “TCP Westwood: congestion window control using bandwidth estimation”. In: *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*. Vol. 3. 2001, 1698–1702 vol.3. DOI: 10.1109/GLOCOM.2001.965869.

[62] Mudassar Ahmad et al. “TCP CUBIC: A Transport Protocol for Improving the Performance of TCP in Long Distance High Bandwidth Cyber-Physical Systems”. In: *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2018, pp. 1–6. DOI: 10.1109/ICCW.2018.8403545.

[63] Dominik Scholz et al. “Towards a Deeper Understanding of TCP BBR Congestion Control”. In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. 2018, pp. 1–9. DOI: 10.23919/IFIPNetworking.2018.8696830.

[64] R. Stewart. “Stream Control Transmission Protocol”. In: *Request For Comments 4960*. 2007, pp. 1–152.

[65] Dr. Vern Paxson, Mark Allman, and W. Richard Stevens. *TCP Congestion Control*. RFC 2581. Apr. 1999. DOI: 10.17487/RFC2581. URL: https://rfc-editor.org/rfc/rfc2581.txt.

[66] Yunhong Gu and Robert L. Grossman. “UDT: UDP-Based Data Transfer for High-Speed Wide Area Networks”. In: vol. 51. 7. USA: Elsevier North-Holland, Inc., May 2007, pp. 1777–1799. DOI: 10.1016/j.comnet.2006.11.009. URL: https://doi.org/10.1016/j.comnet.2006.11.009.

[67] Jana Iyengar and Ian Swett. *QUIC Loss Detection and Congestion Control*. RFC 9002. May 2021. DOI: 10.17487/RFC9002. URL: https://rfc-editor.org/rfc/rfc9002.txt.

[68] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000. URL: https://rfc-editor.org/rfc/rfc9000.txt.

[69] N. Suri et al. “Mockets: a comprehensive application-level communications library”. In: *MILCOM 2005 - 2005 IEEE Military Communications Conference*. 2005, 970–976 Vol. 2. DOI: 10.1109/MILCOM.2005.1605805.

[70] “Bouncy Castle Library”, available online at: https://www.bouncycastle.org/”. In:

[71] Thomas Schucker, Tamal Bose, and Bo Ryu. “Emulating Wireless Networks with High Fidelity RF Interference Modeling”. In: *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 2018, pp. 822–828. DOI: 10.1109/MILCOM.2018.8599761.

[72] Andrei Gurtov et al. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 6582. Apr. 2012. DOI: 10.17487/RFC6582. URL: https://rfc-editor.org/rfc/rfc6582.txt.

[73] *https://en.wikipedia.org/wiki/IPmulticast*. Accessed: 09/18/2021.

[74] Bill Fenner. *Internet Group Management Protocol, Version 2*. RFC 2236. Nov. 1997. DOI: 10.17487/RFC2236. URL: https://rfc-editor.org/rfc/rfc2236.txt.

[75] Jozef Papan, Pavel Segec, and Peter Paluch. “Utilization of PIM-DM in IP fast reroute”. In: *2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 2014, pp. 373–378. DOI: 10.1109/ICETA.2014.7107614.

[76] Tarek R. Sheltami, Anas A. Al-Roubaiey, and Ashraf S. Hasan Mahmoud. “A survey on developing publish/subscribe middleware over wireless sensor/actuator networks”. In: *Wireless Networks* 22 (2016), pp. 2049–2070.

[77] Birman K P. “A Review of Experiences with Reliable Multicast”. In: *Software Practice and Experience* 29 (1999), pp. 741–774.

[78] *NATS messaging system*. https://nats.io. Accessed: 09/18/2021.

[79] *RabbitMQ*. https://www.rabbitmq.com/. Accessed: 09/18/2021.

[80] *AMQP*. https://www.rabbitmq.com/tutorials/amqp-concepts.html. Accessed: 09/18/2021.

[81] *RabbitMQ-JAVA*. https://github.com/rabbitmq/rabbitmq-java-client. Accessed: 09/18/2021.

[82] *MQTT*. https://mqtt.org/. Accessed: 09/18/2021.

[83] Roger A. Light. “Mosquitto: server and client implementation of the MQTT protocol”. In: *Journal of Open Source Software* 2.13 (2017), p. 265. DOI: 10.21105/joss.00265. URL: https://doi.org/10.21105/joss.00265.

[84] *APACHE*. https://www.apache.org/. Accessed: 09/18/2021.

[85] *KAFKA*. https://kafka.apache.org/. Accessed: 09/18/2021.

[86] *Redis*. https://redis.io/. Accessed: 09/18/2021.

[87] *Redis-c*. https://redis.io/topics/cluster-spec. Accessed: 09/18/2021.

[88] *DDS*. https://www.dds-foundation.org/what-is-dds-3/. Accessed: 09/18/2021.

[89] *DDS Specificaiton*. https://www.omg.org/spec/DDS/1.4/PDF. Accessed: 09/18/2021.

[90] *RTPS Specification*. https://www.omg.org/spec/DDSI-RTPS/2.3/Beta1/PDF. Accessed: 09/18/2021.

[91] *DDS Security Specificaiton*. https://www.omg.org/spec/DDS-SECURITY/1.1/PDF. Accessed: 09/18/2021.

[92] *Open DDS*. https://opendds.org/. Accessed: 09/18/2021.

[93] *RTI DDS*. https://www.rti.com/products. Accessed: 09/18/2021.

[94] A. Mazzini et al. "DisService: Network state monitoring and prediction for opportunistic information dissemination in tactical networks". In: *2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*. 2010, pp. 555–560. DOI: 10.1109/MILCOM.2010.5680429.

[95] Filippo Poltronieri et al. "A secure group communication approach for tactical network environments". In: *2018 International Conference on Military Communications and Information Systems (ICMCIS)*. 2018, pp. 1–8. DOI: 10.1109/ICMCIS.2018.8398696.

[96] A. Das, I. Gupta, and A. Motivala. "SWIM: scalable weakly-consistent infection-style process group membership protocol". In: *Proceedings International Conference on Dependable Systems and Networks*. 2002, pp. 303–312. DOI: 10.1109/DSN.2002.1028914.

[97] *ZERO-MQ*. http://zeromq.org/. Accessed: 09/18/2021.

[98] Joseph P. Macker et al. *NACK-Oriented Reliable Multicast (NORM) Transport Protocol*. RFC 5740. Nov. 2009. DOI: 10.17487/RFC5740. URL: https://rfc-editor.org/rfc/rfc5740.txt.

[99] Mark J. Handley and Joerg Widmer. *TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification*. RFC 4654. Aug. 2006. DOI: 10.17487/RFC4654. URL: https://rfc-editor.org/rfc/rfc4654.txt.

[100] *JGROUPS*. http://www.jgroups.org/. Accessed: 09/18/2021.

[101] *https://github.com/edgware/edgware*. Accessed: 09/18/2021.

[102] *The Netherlands Organisation for applied scientific research*. https://www.tno.nl/en/. Accessed: 09/18/2021.

[103] *NATO - AEP-76 VOL IV (RESTRICTED) SPECIFICATIONS DEFINING THE JOINT DISMOUNTED SOLDIER SYSTEM INTEROPERABILITY NETWORK (JDSSIN) - INFORMATION EXCHANGE MECHANISM*. https://standards.globalspec.com/std/14359780/aep-76-vol-iv. Accessed: 09/18/2021.

[104] *NATO - STANAG 4677 - Dismounted Soldier Systems Standards and Protocols for Command, Control, Communications and Computers (C4) Interoperability (DSS C4 Interoperability)*. https://standards.globalspec.com/std/9968493/STANAG\%204677. Accessed: 09/18/2021.

[105] *NATO - STANAG 5527 - FRIENDLY FORCE TRACKING SYSTEMS (FFTS) INTEROPERABILITY*. Accessed: 09/18/2021.

[106] Yannick Lacharite et al. "A Simplified Approach to Multicast Forwarding Gateways in MANET". In: *2007 4th International Symposium on Wireless Communication Systems*. 2007, pp. 426–430. DOI: 10.1109/ISWCS.2007.4392375.

[107] *PF4J*. https://pf4j.org/. Accessed: 10/18/2021.

[108] *TCPDump*. https://www.tcpdump.org/. Accessed: 09/18/2021.

[109] Niranjan Suri et al. "The angloval tactical military scenario and experimentation environment". In: *2018 International Conference on Military Communications and Information Systems (ICMCIS)*. 2018, pp. 1–8. DOI: 10.1109/ICMCIS.2018.8398729.

[110] Nuha Alshuqayran, Nour Ali, and Roger Evans. "A Systematic Mapping Study in Microservice Architecture". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. 2016, pp. 44–51. DOI: 10.1109/SOCA.2016.15.

[111] *SNMP4j*. https://www.snmp4j.org/. Accessed: 10/18/2021.

[112] *SIGAR Library*. https://github.com/hyperic/sigar. Accessed: 10/18/2021.

[113] Martin Révay and Miroslav Líška. "OODA loop in command amp; control systems". In: *2017 Communication and Information Technologies (KIT)*. 2017, pp. 1–4. DOI: 10.23919/KIT.2017.8109463.

[114] Hang Zhang, Dongdong She, and Zhiyun Qian. "Android Root and its Providers". In: Oct. 2015, pp. 1093–1104. DOI: 10.1145/2810103.2813714.

[115] N. Minallah, S. Gul, and M.M. Bokhari. "Performance Analysis of H.265/HEVC (High-Efficiency Video Coding) with Reference to Other Codecs". In: *2015 13th International Conference on Frontiers of Information Technology (FIT)*. 2015, pp. 216–221. DOI: 10.1109/FIT.2015.46.

[116] I. Culjak et al. "A brief introduction to OpenCV". In: Jan. 2012, pp. 1725–1730. ISBN: 978-1-4673-2577-6.

[117] *iPerf3*. https://iperf.fr/. Accessed: 10/18/2021.

[118] Maggie Breedy et al. "Transport protocols revisited". In: *MILCOM 2015 - 2015 IEEE Military Communications Conference*. 2015, pp. 1354–1360. DOI: 10.1109/MILCOM.2015.7357633.

# Acronyms

**ACE** Adaptive Communication Environment

**ACM** Agile Communications Middleware

**AMQP** Advanced Message Queuing Protocol

**API** Application Programming Interface

**AQM** Advanced Queue Management

**BBR** Bottleneck Bandwidth and Round-trip propagation time

**BFD** Blue Force Data

**BRN** Barrage Relay Network

**CACP** Contention–aware Admission Control Protocol

**CB-Slot** Cooperative Broadcast Slot

**CE** CommEffect

**COMSEC** Communication Security Barrier

**COTS** Commercial off-the-shelf

**CTCP** Compound TCP

**cwind** Congestion Window

**DCF** Distributed Coordination Function

**DDS** Data-Distribution Service for Real-Time Systems

**DIL** Delayed/Disconnected Intermittently-Connected Low-Bandwidth

**Doc** HQ Document

**DS** Dissemination Service

**DS-R** DisService without reliability enabled

**DS-R** DisService with reliability enabled

**EMANE** Extendable Mobile Ad-hoc Network Emulator

**GCS** Group Communication Solution

**GDEM** Generic Data Exchange Mechanism

**HOL** Head-Of-Line

**HoLB** Head of Line Blocking

**IHMC** Florida Institute for Human and Machine Cognition

**JDSS** Joint Dismounted Soldier System Information Exchange Mechanism

**LAN** Local Area Network

**LAV** Light Armored Vehicle

**LKSCTP** Linux Kernel Stream Control Transmission Protocol Tools

**MANET** Mobile Ad-Hoc Network

**Mockets** Mobile Sockets

**MPR** Multi Point Relay

**MQTT** MQ Telemetry Transport

**MSS** Maximum Segment Size

**MTU** Maximum Transmission Unit

**NLH** Network Layer Handover

**NMS** Network Management Station

**NORM** NACK-Oriented Reliable Multicast

**OLA** Opportunistic Large Array

**OODA** Observe Orient Decide and Act

**P3** Passive Packet Pair

**PPD** Packet Pair dispersion

**PSCBT** Precomputed Synchronized Cooperative Broadcast

**QoS** Quality of Service

**QUIC** Quick UDP Internet Connections

**ReDiS** Remote Dictionary Server

**RMQ** RabbitMQ

**RPC** Remote Procedure Call

**RTG** Research Task Group

**RTPS** Real-Time Publish-Subscribe (RTPS) Wire Protocol

**RTT** Round Trip Time

**SBT** Shared Based Tree

**SCB** Synchronized Cooperative Broadcast

**SCTP** Stream Control Transmission Protocol

**SD** Sensor Data

**SENSEI** Smart Estimation of Network StatE Information

**SGBT** Shared Group Based Tree

**SLoPS** Self-Loading Periodic Streams

**SMF** Simplified Multicast Forwarding

**SNMP** Simple Network Management Protocol

**SNSE** Smart Network Status Exchange Service

**ssthres** Slow Start Threshold

**STO** Science and Technology Organization

**SUSP** The Simple UDP Segmentation Protocol

**TDMA** Time Division Multiple Access

**TDR** Tactical And Disaster

**TOPP** Trains of Packet Pairs

**TTL** Time to Live

**UAV** Unmanned Aerial Vehicle

**UDT** UDP-based Data Transfer protocol

**VM** Virtual Machine

**VPSP** Variable Packet Size Probing

**ZMQN** ZeroMQ+NORM