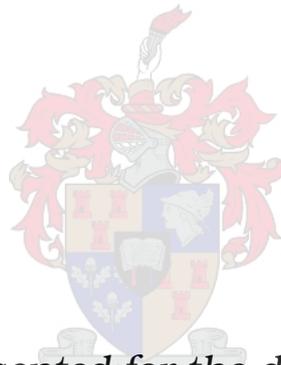


Quantification and Improvement of State Consistency in a Virtual Environment under Adverse Network Conditions

by

Daniël Schoonwinkel



Dissertation presented for the degree of Doctor of Philosophy in Electronic Engineering in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. H.A.E. Engelbrecht

March 2020



UNIVERSITEIT • STELLENBOSCH • UNIVERSITY
jou kennisvenoot • your knowledge partner

Plagiaatverklaring / Plagiarism Declaration

- 1 Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.
- 2 Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
I agree that plagiarism is a punishable offence because it constitutes theft.
- 3 Ek verstaan ook dat direkte vertalings plagiaat is.
I also understand that direct translations are plagiarism.
- 4 Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelike aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
- 5 Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

Abstract

Quantification and Improvement of State Consistency in a Virtual Environment under Adverse Network Conditions

D. Schoonwinkel

*Department of Electrical and Electronic Engineering
University of Stellenbosch
Private Bag X1, Matieland 7602 , South Africa.*

Dissertation: PhD (Eng)

March 2020

Virtual Reality, Augmented Reality and Virtual Environments (VE) all share a critical requirement: the virtual environment needs to be consistent and interactive, in order to provide the realism that enables a user to be immersed in an environment other than their own.

State in Virtual Environment is the information that needs to be disseminated to every user's interface in order to render the virtual environment. Ensuring that each user has a live and consistent view of the VE, requires that each user receives the most recent changes to the VE. Interactivity in the VE increases the amount of information that must be disseminated between interacting users, and also means that updates must be sent with low latency to ensure state consistency. Furthermore, in order to complete the immersion into the virtual world, the user must be disconnected from her current world: by using wireless communication, the user can be untethered and free to move around to experience the virtual world. Wireless virtual reality is thus seen as a necessity for virtual immersion.

As the next generation of mobile wireless communication, the 5G specification promises high throughput of up to 10Gbps data rate, 1 millisecond latency and ultra-high reliability coupled with a 100-fold increase in the number of connected devices. 5G technology could finally enable distributed, fully immersive, wireless Virtual Reality. Wireless communication is susceptible to interference and therefore

prone to information loss. In order to understand how wireless loss will affect state consistency, we need to evaluate state degradation under adverse network conditions, such as loss, network delay and bandwidth limitations.

In this dissertation we quantify the state consistency by using VAST, an existing VE implementation, and the Mininet network emulator, which allows controlling different adverse network conditions. We found that under most adverse network conditions, the state consistency degrades significantly. We also found that UDP was less sensitive to adverse conditions than TCP.

In order to improve the state consistency, we propose a network coding based packet loss mitigation protocol that uses the UDP transport protocol. We term this new protocol UDPNC and show that it is successful at improving state consistency and also performs well when network delay is introduced. However, we show that UDPNC requires significantly more bandwidth than other network transport protocols and suffers in scenarios where bandwidth is limited. However, assuming that a high bandwidth wireless connection is available, such as provided by 5G mobile or Wi-Fi 802.11ac, we argue that the extra bandwidth would not present a significant cost to improving state consistency.

Uittreksel

Quantification and Improvement of State Consistency in a Virtual Environment under Adverse Network Conditions

D. Schoonwinkel

*Department of Electrical and Electronic Engineering
University of Stellenbosch
Private Bag X1, Matieland 7602, South Africa.*

Proefskrif: PhD (Eng)

Maart 2020

Virtuele realiteit, toegevoegde realiteit en virtuele omgewings het 'n kritiese vereiste in gemeen: die virtuele omgewing moet konsekwent en interaktief bly om die gebruiker in die realisme van die virtuele omgewing in te dompel.

Die toestand van die Virtuele Omgewing (VO) is die inligting wat versprei moet word na elke gebruiker se toestel om die omgewing te vertoon. Om te verseker dat elke gebruiker 'n aktuele en konsekwente beeld van die VO het, vereis dat elke gebruiker die mees onlangse veranderinge aan die VO ontvang. Interaktiwiteit in die VO vermeerder die hoeveelheid inligting wat versprei moet word na gebruikers. Hierdie inligting moet ook met 'n lae wagtyd versprei word om die virtuele toestand konsekwent te hou. Om die indompeling van die virtuele omgewing verder te vervolmaak, moet die gebruiker ontkoppel word van die vereistes van die huidige wêreld: deur die gebruik van draadlose kommunikasie kan die gebruiker vrylik in die virtuele omgewing rond beweeg. Draadlose virtuele realiteit is dus noodsaaklik vir volledige indompeling.

5G kommunikasie is die nuwe generasie draagbare-draadlose spesifikasie wat hoë bandwydte van tot 10 Gbps, 1 millisekonde wagtyd en ultra-betroubare kommunikasie waarborg. 5G kommunikasie sal ook 'n honderdvoudige vermeerdering in die aantal gekonnekteerde toestelle bewerkstellig. 5G tegnologie kan uiteindelik ver-

spreide volledig-indompelende en draadlose Virtuele Realiteit toepassings bewerkstellig.

Draadlose kommunikasie is vatbaar vir interferensie en dus geneig tot inligting verlies. Om te verstaan hoe draadlose inligting verlies die konsekwentheid van die virtuele omgewing beïnvloed, moet ons die agteruitgang onder ongunstige netwerk toestande beoordeel. Ongunstige netwerk toestande sluit in inligting verlies, netwerk vertraging en bandwydte beperkings.

In hierdie proefskrif kwantiseer ons die toestand konsekwentheid deur om van VAST, 'n bestaande VO implementasie, gebruik te maak en die Mininet netwerk emuleerder te gebruik om die ongunstige netwerk toestande te beheer. Ons het gevind dat onder meeste ongunstige netwerk toestande die konsekwentheid van die virtuele omgewing verswak. Ons het ook gevind dat UDP minder sensitief is tot ongunstige toestande in vergelyking met TCP.

Om die konsekwentheid van die virtuele omgewing te verbeter, stel ons 'n netwerk-kodering gebaseerde pakkie-verlies-vermindering protokol voor wat die UDP protokol gebruik. Ons noem hierdie protokol UDPNC en bewys dat dit suksesvol is om konsekwentheid in die VO te verbeter. Ons bewys ook dat UDPNC korrek funksioneer en die konsekwentheid behou selfs al word pakkies in die netwerk vertraag. Ons het egter gevind dat UDPNC aansienlik meer bandwydte vereis as ander netwerk protokolle en verloor effektiwiteit onder bandwydte beperkings.

Indien ons egter aanvaar dat 'n hoë bandwydte konneksie soos 5G of Wi-Fi 802.11ac gebruik sal word, beweer ons dat die ekstra bandwydte nie 'n noemenswaardige koste sal wees om die toestand-konsekwentheid van die VO te verbeter nie.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- My lecturer, Prof Herman Engelbrecht for asking the right questions and believing in me when I did not.
- The Medialab and all my friends and colleagues.
- My older brother, Johan Schoonwinkel for excellent, patient and continued on help with C++ programming and debugging.
- My significant other, Irene van Staden for support, pep-talks and always believing I can finish it in time.
- My family and friends for many hours of praying and encouraging me.
- For Martin Molin from the Youtube channel *Wintergatan* for inspiring me to try again when I failed, just as he tried again every time.

Dedication

Aan my Hemelse Vader. Ek is tot alles in staat deur Hom wat my krag gee.

Contents

Abstract	ii
Uittreksel	iv
Acknowledgements	vi
Dedication	vii
Contents	viii
List of Figures	xii
List of Tables	xviii
Nomenclature	xxii
1 Introduction	1
1.1 Motivation	1
1.2 Fully Immersive Virtual Reality	1
1.3 Wireless VR	3
1.4 Network Coding on Wireless Networks	3
1.5 Problem Statement	4
1.6 Research Questions	4
1.7 Contributions	4
1.8 Research Approach	5
1.9 Dissertation layout	6
2 State Consistency	7
2.1 Distributed State	7
2.2 Architectures	11
2.3 Scalability Strategies	18
2.4 Communication Strategies	20
2.5 State Management Strategies	26
2.6 Related research: VE implementations	28
2.7 Why Spatial Publish/Subscribe and Voronoi Self-Organising?	34

2.8	VAST Library	36
2.9	VAST as a Case Study	38
2.10	Application Domain	39
2.11	Summary	39
3	Adverse Network Conditions on State Consistency	40
3.1	Adverse Network Effects Definition	40
3.2	Effect of Adverse Network Conditions on State Consistency	43
3.3	TCP and UDP	45
3.4	Summary	46
4	Results of VAST MMVE Under Adverse Network Conditions	47
4.1	Mininet Emulator and POX Controller	47
4.2	Network Setup	49
4.3	Additions to VAST	50
4.4	Real versus Simulated Network Stack	50
4.5	Consistency Metrics	51
4.6	Network Performance Metrics	54
4.7	VAST Components	55
4.8	Simulation Parameters	58
4.9	Nominal Network Conditions	59
4.10	Adverse Network Conditions Testing	62
4.11	Packet Loss Tests	63
4.12	Comparing Mininet Hosts with Physical Hosts	79
4.13	Delay Tests	84
4.14	Bandwidth Limited Tests	87
4.15	Scalability Tests	91
4.16	General discussion and recommendations	95
4.17	Which problem can be addressed on Transport Layer?	96
4.18	Summary	96
5	Mitigation Strategies for Packet Loss	97
5.1	Open Systems Interconnect (OSI) Model	97
5.2	Communicating Using a Network Stack	98
5.3	The status quo: What happens in networks currently?	99
5.4	Recovery and Compensation Mechanisms (i.e. how can we do better?)	100
5.5	Requirements And Considerations of Packet Loss Mitigation Strategy	105
5.6	Naive Erasure Recovery	106
5.7	Erasure Codes	106
5.8	Network Coding	109
5.9	Cost of Using Intra-flow Network Coding	119

5.10	Network Coding as Packet Loss Mitigation Strategy . . .	122
5.11	Other Applications of Network Coding	122
5.12	Summary	125
6	Implementing Network Coding in VAST	126
6.1	Chapter layout	126
6.2	Network Coding Network Setup	127
6.3	Inter-flow and Intra-flow NC for Interactive Applications	127
6.4	Network Coding Strategy	129
6.5	Encoding Process	132
6.6	Decoding Process	136
6.7	Fulfilling Design Requirements and Considerations . . .	143
6.8	Summary	144
7	Results of Using Network Coding as Packet Loss Mitigation Strategy	145
7.1	Network Coding Network Setup	145
7.2	Simulation Parameters	145
7.3	VAST Components	145
7.4	Nominal test	147
7.5	Why we only look at UDP vs UDPNC	150
7.6	Packet Loss Mitigation Evaluation and Adverse Network Conditions Testing	150
7.7	Packet Loss Tests	151
7.8	Comparing Mininet Hosts with Physical Hosts	160
7.9	Delay Tests	164
7.10	Discussion	165
7.11	Bandwidth Limited Tests	165
7.12	Scalability Tests	170
7.13	Cost of UDPNC	174
7.14	Emulated Network Model Inaccuracies and Consequences	174
7.15	Benefits of Mininet Emulated Network	175
7.16	Summary	175
8	Conclusions and Future Work	177
8.1	How Does Adverse Network Conditions Affect The State Consistency of a Virtual Environment?	177
8.2	Is Network Coding an Effective Packet Loss Mitigation Strategy?	178
8.3	VAST Case Study and Limitations	180
8.4	Comparison to Prior Work	181
8.5	Concluding Remarks	184
8.6	Future Work	184
8.7	Future Research Questions	187

Bibliography	189
Appendices	202
A Test Data Acquisition	203
A.1 Movement and node status: VASTStatLog	203
A.2 Latency: VASTLatencyStatLog	204
A.3 NIC Send / Receive bandwidth: tshark	205
A.4 Network Coded Bandwidth: VASTNetStatLog	205

List of Figures

2.1	Direct Connection between peers. The inner red circle represents the client and the outer circle the client's AOI. Blue arrows indicate connections.	20
2.2	Forwarding among neighbours.	22
2.3	Spanning tree forwarding. Source: [44]	23
2.4	Multicast groups.	24
2.5	Spatial publish subscribe.	26
2.6	Apolo connections. Source: [82].	30
2.7	Real-Time Framework (RTF) interactions. Source: [100].	33
2.8	S-VON architecture, relays represent super-peers with connected clients, <i>Source: [71]</i>	38
4.1	Mininet network setup using a POX controller.	48
4.2	VAST and NIC bandwidth comparison with no packet loss. The black line indicates the end of the setup phase, meaning that the network has reached steady state. From this point onwards, all the nodes in the VE are in the <i>JOINED</i> state. As there is no packet loss in this run, no change in network conditions are applied. Averages of the setup phase are given on the left, and averages after the setup phase are given on the right.	56
4.3	VAST and NIC bandwidth comparison with 10% packet loss. The black line indicates the end of the setup phase, meaning that the network has reached steady state. From this point onwards, all the nodes in the VE are in the <i>JOINED</i> state. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. Averages before packet loss are given on the left and averages after packet loss is applied are given on the right.	57
4.4	Nominal network conditions. Aggregate topology consistency, normalised drift distance, average latency, and normalised NIC send and receive bandwidths are shown for TCP (blue) and UDP (red) tests. Black circles indicate outliers.	61

- 4.5 Results of a TCP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. In this figure the drift distance plot is scaled to the range of normalised drift distance for readability. See Figure 4.6 for full range of maximum drift distance. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. 65
- 4.6 Results of a TCP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Topology consistency and drift distance plot shows the full range. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. . . . 66
- 4.7 Results of a UDP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. In this figure the drift distance plot is scaled to the range of normalised drift distance for readability. See Figure 4.8 for full range of maximum drift distance. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. 69

4.8	Results of a UDP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Topology consistency and drift distance plot shows the full range. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. . . .	70
4.9	Results of a 30% packet loss TCP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Topology consistency and drift distance plot shows the full range. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right.	73
4.10	Aggregated summary of 20 runs, each with 50 nodes. TCP results shown in blue and UDP in red. Outliers are indicated with black circles. Only packet loss percentages up to 10% is shown in this graph to facilitate readability. Full results are shown in Tables 4.3,4.4.	74
4.11	Mininet hosts versus physical hosts using TCP. The Mininet performance is shown in blue and the physical network performance is shown in black. Outliers are indicated as black circles.	82
4.12	Mininet hosts versus physical hosts using UDP. The Mininet performance is shown in red and the physical network performance is shown in black. Outliers are indicated as black circles.	83
4.13	Network delay of 0 - 200 ms. TCP results shown in blue and UDP in red. Outliers are indicated with black circles. .	86
4.14	Bandwidth limitation tests. The x-axis indicates an increasingly strict limitation on the bandwidth. Outliers are indicated with black circles. Only the range of 1 MBps is shown as nominal state consistency was observed for higher allowed bandwidth. Table 4.21 shows all of the results of the bandwidth test.	90

4.15	Scaling test with 20 - 100 nodes. TCP results are shown in blue and UDP results in red. All tests were run at a packet loss percentage of 10% and zero delay. Outliers are indicated with black circles.	93
5.1	OSI model and Internet Protocol suite abstraction layers.	98
5.2	Network Coding in the Butterfly Network, <i>Source: [59]</i>	110
5.3	Wireless Network Coding, <i>Source: [59]</i>	112
5.4	End-to-End Coding, <i>Source: [115]</i>	112
5.5	Hop-by-Hop Coding, <i>Source: [115]</i>	113
5.6	Randon Linear Network Coding, <i>Source: [115]</i>	113
6.1	Network coding system setup.	128
6.2	Filling packet pools. The circles indicate the network switch or wireless router. The red arrows with a cross indicate a failed transmission. All sent packets are also added to the local packet pool. All UDP packets on the network are also routed to the coding host.	130
6.3	Decoding with packet pools. The circles indicate the network switch or wireless router. Packet (x_1+x_2) is generated by the coding host as a redundant packet. It is forwarded via the router to both recipients that can possibly decode it. Using the stored packets, the other packet can be decoded.	131
6.4	The UDPNC Header. The shown data types are defined as follows: <code>byte:unsigned</code> 1 byte, <code>packetid_t: unsigned</code> 8 bytes, <code>id_t: unsigned</code> 8 bytes, <code>Vast::IPaddr:</code> 4 byte IP address, 2 byte port number, and 2 padding bytes, total of 8 bytes. The checksum field is an unsigned 4 byte integer. The data field is a variable length field based on the packet size field.	132
6.5	Coding process	135
6.6	Uncoded packet pipeline	137
6.7	Coded packet pipeline	138
6.8	Order input	141
7.1	Network coding system setup.	146
7.2	Comparison of TCP, UDP and UDPNC under nominal conditions. Aggregate topology consistency, normalised drift distance, average latency, and normalised NIC send and receive bandwidths are shown for TCP (blue), UDP (red) and UDPNC (green) experiments. Black circles indicate outliers.	148

- 7.3 Results of a UDPNC run showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase, also marked as point c). Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. In this figure the drift distance plot is scaled to the range of normalised drift distance for readability. Points a) and b) indicate anomalies in drift distance, latency and NIC receive bytes due to packet pool clearing. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. See Figure 7.4 for full range of maximum drift distance. 154
- 7.4 Results of a UDPNC run showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. Drift distance plot now shows the full range of maximum drift distance. 155
- 7.5 Aggregated summary of 20 runs, each with 50 nodes. UDP results shown in red, UDPNC in green. Outliers are indicated with black circles. Only packet loss percentages up to 10% is shown in this graph to facilitate readability. Full results are shown in Tables 7.4, 7.5 as well as Figure 7.6. 157
- 7.6 Aggregated summary of 20 runs, each with 50 nodes. UDP results shown in red, UDPNC in green. Outliers are indicated with black circles. The entire range of 0-70% packet loss rates is shown. 158
- 7.7 Mininet hosts versus physical hosts packet loss test using UDPNC. The Mininet performance is shown in green and the physical network performance is shown in black. Outliers are indicated as black circles. 163
- 7.8 Delay tests. Test was run at 10% packet loss with 0 - 200 ms delay. UDP is shown in red and UDPNC in green. Outliers are indicated with black circles. 166

7.9 Bandwidth limitation tests. UDP results are shown in red and UDPNC in green. The x-axis indicates an increasingly strict limitation on the bandwidth. Outliers are indicated with black circles.	169
7.10 Scaling test with 20 - 100 nodes. UDP results are shown in red and UDPNC results in green. All tests were run at a packet loss percentage of 10% and zero delay. Outliers are indicated with black circles.	172

List of Tables

2.1	Architecture comparisons based on Schiele et al's [107] requirements.	17
2.2	Summary of VE implementations	35
4.1	General simulation parameters	58
4.2	Packet loss test simulation parameters	64
4.3	Median of average topology consistencies with 90% confidence interval, for increasing packet loss test.	68
4.4	Medians of normalized drift distances [VE units] with 90% confidence interval, for increasing packet loss test.	71
4.5	Medians of normalized drift distances [VE units] with degradation percentages, for increasing packet loss test.	71
4.6	Median of average latencies [ms] with 90% confidence interval, for increasing packet loss test.	72
4.7	Mean and standard deviation of 100 round-trip ping times on Mininet and physical network. Results in microseconds.	80
4.8	Comparison test simulation parameters	80
4.9	Median of normalised drift distances [VE units] and 90% confidence intervals for TCP test when comparing Mininet and physical hosts.	81
4.10	Median of average latencies [ms] and 90% confidence intervals for TCP test when comparing Mininet and physical hosts.	81
4.11	Median of normalised drift distances [VE units] and 90% confidence intervals for UDP test when comparing Mininet and physical hosts.	81
4.12	Median of average latencies [ms] and 90% confidence intervals for UDP test when comparing Mininet and physical hosts.	81
4.13	Delay test simulation parameters	85
4.14	Medians of normalized drift distances [VE units] with 90% confidence interval when adding network delay.	85
4.15	Median of normalized drift distances [VE units] with degradation percentages when adding network delay.	85

4.16	Median of average latencies [ms] with 90% confidence interval when adding network delay.	85
4.17	Median of normalised NIC send bandwidth [kBps/node], 90% confidence intervals and total bandwidth of the network when adding network delay.	86
4.18	Bandwidth limit simulation parameters	88
4.19	Median of normalised drift distances [VE units] and 90% confidence interval when limiting transmission bandwidth.	88
4.20	Median of normalised drift distances [VE units] with degradation percentages when limiting transmission bandwidth.	89
4.21	Median of average latencies [ms] and 90% confidence interval when limiting transmission bandwidth.	89
4.22	Median of normalised NIC receive bandwidth [kBps/node], 90% confidence intervals and total bandwidth of the network when limiting transmission bandwidth.	89
4.23	Scalability test simulation parameters	91
4.24	Median of normalised drift distances [VE units] and 90% confidence interval when increasing the number of nodes.	92
4.25	Median of average latencies [ms] and 90% confidence interval when increasing the number of nodes.	92
4.26	Median of normalised NIC send bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when increasing the number of nodes.	94
4.27	Median of normalised drift distances [VE units] and 90% confidence interval when increasing the number of nodes.	94
4.28	Median of normalised VAST send bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when increasing the number of nodes.	95
5.1	Summary of Mitigation Strategies	105
7.1	General simulation parameters	147
7.2	Medians and 90% confidence interval of state consistency and network metrics for nominal test.	148
7.3	Packet loss test simulation parameters	152
7.4	Median of average topology consistencies with 90% confidence interval when increasing packet loss.	153
7.5	Medians of normalized drift distances [VE units] with 90% confidence interval when increasing packet loss.	156
7.6	Medians of normalized drift distances [VE units] with degradation percentages when increasing packet loss.	156
7.7	Median of normalised NIC receive bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when increasing packet loss.	159

7.8	Mean and standard deviation of 100 round-trip ping times on Mininet and physical network. Results in microseconds.	161
7.9	Comparison test simulation parameters.	162
7.10	Median of normalised drift distances [VE units] and 90% confidence interval for UDPNC test when comparing Mininet hosts to physical hosts.	162
7.11	Median of average latencies [ms] and 90% confidence interval for UDPNC test when comparing Mininet hosts to physical hosts.	162
7.12	Delay test simulation parameters	165
7.13	Median of average latencies [ms] and 90% confidence interval when adding network delay.	165
7.14	Bandwidth limit simulation parameters	166
7.15	Median of average topology consistencies with 90% confidence interval when limiting bandwidth.	167
7.16	Medians of normalized drift distances [VE units] with 90% confidence interval when limiting bandwidth.	167
7.17	Medians of normalized drift distances [VE units] with degrade percentages when limiting bandwidth.	167
7.18	Median of average latencies [ms] and 90% confidence interval when limiting bandwidth.	168
7.19	Median of normalised NIC receive bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when limiting bandwidth.	168
7.20	Scalability test simulation parameters	170
7.21	Median of average topology consistencies with 90% confidence interval, for increasing packet loss test.	171
7.22	Medians of normalized drift distances [VE units] with 90% confidence interval when scaling number of nodes.	171
7.23	Median of average latencies [ms] and 90% confidence interval when scaling number of nodes.	171
7.24	Median of normalised NIC send bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when scaling number of nodes.	173
8.1	Summary of drift distance and degradation percentages under adverse conditions.	177
8.2	Summary of best topology consistency and drift distance UDPNC mitigation under packet loss.	179
8.3	Summary of related work on VAST	181
8.4	Summary of experience latency under adverse network conditions compared to related work.	182
8.5	Summary of network delay results.	183

A.1 Mean and standard deviation of 100 round-trip ping times
on physical network. Results in milliseconds. 204

Nomenclature

Abbreviations

ACI	Adjacent Channel Interference
ACK	Acknowledgment
ALM	Application Layer Multicast
AOI	Area-of-Interest
AR	Augmented Reality
ARP	Address Resolution Protocol
ARQ	Automatic Retransmission Request
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CoAP	Constrained Application Protocol
CTS	Clear-To-Send
DARPA	Defense Advanced Research Projects Agency
DASH	Dynamic Adaptive Streaming of HTTP
DCF	Distributed Coordination Function
DHT	Distributed Hash Table
DIFS	Distributed Coordination Function Inter-Frame Space
DoS	Denial-of-Service
E2E	End-to-End
ECC	Error Correction Codes
FPS	First-Person Shooter
GAS	Greedy Anchor Select
HbH	Hop-by-Hop
HMD	Head Mounted Display
HPS	Human Patient Simulator
HTTP	Hyper-Text Transfer Protocol
IoT	Internet-of-Things
IP	Internet Protocol
IQR	Interquartile range

ISM	Industrial, Scientific and Medical
LDPC	Low-Density Parity Check
LT	Luby Transforms
MAC	Medium Access Control
MCS	Modulation and Coding Scheme
MEC	Mobile Edge Computing
MMOG	Massively Multiplayer Online Game
MMVE	Massively Multi-user Virtual Environment
MQTT	MQ Telemetry Transport
MTU	Maximum Transmission Unit
NACK	Not Acknowledgment
NC	Network Coding
NIC	Network Interface Card
NTP	Network Time Protocol
OSI	Open Systems Interconnect
P2P	Peer-to-Peer
PAFV	Peer Assisted Freeview Video
POX	Python based Network Controller
PPR	Partial Packet Recovery
Q1	First quartile
Q3	Third quartile
QoE	Quality of Experience
QoS	Quality of Service
QUIC	UDP-based Transport Protocol
RAS	Random Anchor Select
RLNC	Random Linear Network Coding
RTF	Real-Time Framework
RTO	Retransmission Time-out
RTS	Request-to-Send
RTT	Round-trip Time
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Network
SPS	Spatial Publish Subscribe
S-VON	Spatial Publish Subscribe on Voronoi Overlay Network
TCP	Transport Control Protocol
VAST	VON-based Application-layer Spatial publish / subscribe Topology awareness

VE	Virtual Environment
VoIP	Voice-over-IP
VON	Voronoi Overlay Network
VR	Virtual Reality
VSO	Voronoi Self-Organising

Units

Bps	Bytes per second
kBps	Kilobytes per second
MBps	Megabits per second
ms	Milliseconds

Term Definitions

User	A user (person) of the virtual environment.
Client	A Client entity representing the user in the VE. Clients receive game updates.
Neighbour	A client entity in close proximity to another client.
Host	A computer capable of running software and communicating on the network.
Node	A host running the virtual environment software.

State Consistency Metrics

Topology Consistency

$$TC\% = \frac{\sum_{k=1}^K N_k}{\sum_{k=1}^K M_k} \times 100 \quad (1)$$

where K is the total number of nodes in the VE, N_k is the known AOI neighbours and M_k is the actual neighbours at node k that falls within its AOI.

Normalised Drift Distance

$$NDD = \frac{\sum_{k=1}^K \sum_{n=1}^N (\text{dist}(PP_{n,k}, AP_{n,k}))}{\sum_{k:DD_k \neq 0}^K 1} \quad (2)$$

where N is the known AOI neighbours, dist is the Euclidean distance function in a 2D plane rounded to the nearest integer, $PP_{n,k}$ the perceived position of the neighbour n at node k , $AP_{n,k}$ the actual position of the neighbour and K is all the nodes in the network.

$\sum_{k:DD_k \neq 0}^K 1$ counts the number of nodes experiencing drift at the current step. The unit for this measurement is in-game world distance units.

Average VAST Latency

$$L = \frac{\sum_{i=1}^R (t_r - t_s)}{R} \quad (3)$$

Where L is the latency, t_s the clock time at message creation and t_r the clock time when receiver processes the message. R is the number of latency records.

Normalised VAST Send / Receive Bandwidth

$$\text{NVBW} = \frac{\sum_{i=1}^K \text{VBW}_i}{K} f_{\text{step}} \quad (4)$$

where VBW_i is the VAST send or receive bandwidth of node i at the current step, K the number of nodes and f_{step} the update frequency.

Normalised NIC Send / Receive Bandwidth

$$\text{NNBW} = \frac{\sum_{i=1}^K \text{NBW}_i}{K} f_{\text{step}} \quad (5)$$

where NBW_i is the NIC send or receive bandwidth of node i at the current step, K the number of nodes and f_{step} the update frequency.

Chapter 1

Introduction

1.1 Motivation

With the introduction of 5G of mobile broadband, low latency and ultra high reliability, fully immersive, interactive and distributed Virtual Reality (VR) and Augmented Reality (AR) could finally be achievable in the near future. According to Elbamy et al. [53] mmWave technologies and mobile edge computing (MEC) could, with some improvement, scalability considerations and careful network integration, enable the 1 ms latency and 20 Gbits/s throughput. Ultra high reliability is proposed to be achieved with multi-connectivity over the same interface or different wireless technologies (eg. mobile wireless and Wi-Fi). Firstly we will explain the concept and requirements of fully immersive Virtual Reality.

1.2 Fully Immersive Virtual Reality

The first Virtual Reality (VR) Head Mounted Display (HMD) was developed in 1968 by Ivan Sutherland and Bob Sproul [113]. However, it is only in the past decade that VR became a commercial reality, starting with the Oculus Rift prototype developed by Palmer Lucky in 2011 [20]. A significant amount of work has gone into improving headset hardware, in particular to reduce motion blur and picture latency [105]. This is required for the experience to be realistic and prevent motion sickness during use.

Currently there are two approaches of VR HMDs: stand-alone and tethered devices [19]. Stand-alone devices (for example the *Oculus Quest* [11]) allows the user to use the entire physical space without the need for extra equipment. It has its own battery, processor and storage, allowing the user to experience VR by installing applications on the device. However, due to its mobile nature, it is

limited by its processing power and battery life (2 - 3 hours). In contrast, the *Valve Index* [99] has wires connected to a PC for display and inputs. Controllers are connected to the headset wirelessly and user inputs are sent to the PC via USB 3.0. The image processing is done on the PC, enabling more performance than stand-alone versions with higher resolution and better refresh rates. However, the physical space use is limited to cable length, typically 5 meters. To summarise: stand-alone devices provide freedom of movement, but lack graphics processing power, high bandwidth communication and battery life while tethered devices limit the user's movement and multi-user support (with cables becoming tangled for example).

Although VR has come a long way, further improvements are still needed to provide a *fully immersive* VR experience. Bastug, Benis, Medard and Debbah [33] described a VR system as *fully immersive* when the client cannot distinguish between the virtual and real world experiences. Fully immersive VR is a unique use-case that sets ambitious requirements on bandwidth, latency and reliability. Bastug et al. estimate that a network throughput of at least 5.2 Gb/s is required to render 360° video at 30K resolution. Furthermore, latency must be below 13 ms [102] to be imperceptible to the human eye.

As the popularity of multiplayer VR applications grows, such as Facebook's *Horizon* [4] massively multiplayer virtual world, fully immersive systems will require scalable solutions for handling many players.

According to Westphal [118], most VR (and augmented reality (AR)) research focus is placed on the display and potential applications, with little attention given to the networking requirements. Westphal states that the network needs to provide an updated representation of the Virtual Environment (VE) underlying the VR experience. If the network does not fulfill the above-mentioned requirements, the VE representation will be incomplete and it will be perceptible in the VR experience. Westphal mentions that, especially in a VR gaming application, the interactions between players will need a strong network foundation to maintain a consistent state (view of the VE) between players. Westphal suggests that 5G technologies such as Mobile Edge Computing (MEC) and higher mobile network speeds can help satisfy these requirements.

In an article focused on VR medical training [65], Hamza-Lup, Rolland and Hughes implement an AR training prototype for performing endotracheal intubations (frequently performed by paramedics) on a Human Patient Simulator (HPS) dummy. The prototype allows trainees to perform the procedure on the HPS with an AR overlay (seen through an HMD) showing the internal organs. This AR overlay

can also be observed by an instructor and other students simultaneously. The authors refer to this AR overlay as the *dynamic shared state* which needs to be kept consistent between all the participants, especially the instructor, to allow real-time feedback on the trainee's progress. They propose a synchronisation algorithm to ensure that all viewing parties will be synchronised at least at the speed of the state updates. In that paper's setup only the main actor's perspective and a visual overlay is streamed; other trainees are unable to interact except commentary on a shared voice channel. It is therefore not truly an interactive VR/AR system, but demonstrates how VR/AR is also a state consistency problem.

Fully immersive VR is interactive and therefore reliability is a major concern. According to Elbamby, Perfecto, Bennis and Doppler [53] even if latency and bandwidth requirements can be met, the network should ultimately be reliable to maintain the live feel of the virtual world. Any unreliability will cause inconsistencies in the VE and will degrade the VR experience.

1.3 Wireless VR

Chen, Saad and Yin [46] states that only *wireless* VR will be fully immersive as it untethers the user from the real world. Abari, Bhargava, Duffield and Katabi [28] presents a possible solution for wireless communications capable of VR data-rates by using the 24 GHz (mmWave) ISM band. From these two papers it is clear that wireless is necessary and possible for VR.

However, wireless communication is generally a lossy physical layer and will therefore adversely affect the state consistency of the VE. An explanation of why this wireless is lossy is beyond the scope of our work, but reasons include interference from other wireless devices [123], the hidden node problem [38] as well as adjacent channel interference [112].

In summary: for fully immersive VR, we will need wireless communications and a consistent view of the VE. However, lossy wireless communication could degrade the state consistency and thus the VR experience.

1.4 Network Coding on Wireless Networks

Network coding was introduced in 2000 by Ahlswede et al. [30] by demonstrating a network scenario, the now famous *Butterfly Network*, in which encoding packets together *within* the network can

outperform normal packet forwarding techniques. In a paper by Katti, Muriel and Crowcroft [77] they show even better throughput improvements by using network coding on a wireless mesh network. They conclude that this improvement is due to network coding's ability to better utilise the shared broadcasting channel.

Prior and Rodrigues [103] use network coding in content distribution to generate redundant packet streams, enabling packet loss recovery and thereby improving reliability. In another paper by Zhang, Liu, Chan, and Cheung [124] network coding was used in Free Viewpoint Video streaming to increase the viewpoints available to the user by allowing the exchange of coded viewpoints among each other. Free viewpoint video is similar to VR in that it also provides a 360-degree view, although the viewpoint is stationary in the virtual environment.

The suitability of network coding to wireless networks and the potential in an interactive applications such as free viewpoint video, motivated us to consider network coding as a reliability mechanism in our virtual environment application.

1.5 Problem Statement

The problem statement of this dissertation is as follows:

How can the state consistency of an interactive application such as a Massively Multiplayer Virtual Environment (MMVE) or shared Virtual Reality environment be maintained under adverse (wireless) network conditions?

1.6 Research Questions

From the above problem statement we formulate two research questions:

1. How does adverse network conditions affect the state consistency of a Virtual Environment (VE)?
2. Is Network Coding an effective packet loss mitigation strategy?

1.7 Contributions

In this dissertation we make the following contributions:

1. We quantify the adverse effects of packet loss, network delay, and bandwidth limitations on the state consistency of an inter-

active application. We achieve this by applying adverse network conditions to an emulated network and measuring state consistency metrics of a virtual environment.

2. We propose a novel network coding strategy which will improve communication reliability without adding significant processing delay.
3. We test the proposed network coding strategy and show improved state consistency in the virtual environment.

1.8 Research Approach

In order to answer our two research questions, we present the dissertation in two parts. Firstly we will measure the effect of adverse network conditions. In the second part, we will investigate network coding as a possible *packet loss* mitigation strategy.

1.8.1 Measuring State Consistency

We perform tests under different packet loss, network delay and bandwidth limit conditions and measure the state consistency with the following metrics.

Topology consistency is the ratio of the number of known client neighbours divided by the number of true client neighbours in the VE. This metric describes the cohesion of the VE.

Drift distance is the difference between the actual and perceived position of a neighbour node.

In the VE a larger drift distance indicates a lack of accuracy in neighbour positions, typically due to packet loss or delay in communication. Lower topology consistency means that the nodes are completely unaware of their neighbours, also due to adverse network conditions. Topology consistency and drift distance are related in the sense that a high drift distance will typically accompany a lower topology consistency. However, topology consistency is a more lenient consistency metric whereas drift distance is very sensitive to immediate differences in state.

In addition to the state consistency metrics, we measure client latency and application bandwidth usage. **Client latency** is the time taken for a message to travel through the network and be processed by the receiving client (neighbour). **Application bandwidth** is the number of bytes sent and received in order to transmit the client messages.

1.8.2 Network Coding as Mitigation Strategy

In the second part of the dissertation we will investigate the efficacy of using network coding to increase the reliability of communications and thereby increase the state consistency. We describe network coding in general as well as how it was implemented in our application. We will run the tests in the same conditions as in the first part of the dissertation and compare the results with and without the mitigation strategy.

In the next section we will discuss the detail layout of the dissertation. Chapters 2 to 4 concerns the first research question and chapters 5 to 7 concerns the second research question.

1.9 Dissertation layout

In chapter 2 we will explain the state consistency problem and how it has been addressed in related work.

In chapter 3 we will explain the possible adverse network conditions that a wireless, distributed VE system could experience. We will also discuss UDP and TCP, frequently used protocols for transporting data.

In chapter 4 we present our test setup, how we will be quantifying state consistency and give state consistency results of an MMVE system under different adverse network conditions.

In chapter 5 we will present the Open System Interconnect (OSI) model and discuss available recovery methods on different layers of the OSI model. We explain network coding and applications already using network coding. We explain how network coding can be used as a mitigation strategy in our scenario and the cost in bandwidth that network coding incurs.

In chapter 6 we explain the details of our network coded UDP implementation.

In chapter 7 we present the results of running the state consistency tests with network coded enabled UDP.

Finally, we draw conclusions in chapter 8 from the results in chapter 4 and chapter 7. We also present recommendations and considerations for developers of interactive virtual environment platforms in adverse network conditions. We also discuss possible future work.

Chapter 2

State Consistency

In this chapter we will discuss the distributed state consistency problem in a Virtual Environment (VE).

2.1 Distributed State

In order for users to perceive a VE, a representation of the VE is needed on their local terminal. This representation can then be rendered to a display. We will refer to this representation as the state of the VE. If the user wants to manipulate the VE, they can do so by taking an action on their terminal, typically with a mouse and keyboard or other controllers. In augmented reality systems the state is also associated with the real environment around the user and could also possibly be changed due to an action in the real world. As long as only a single user is using the virtual or augmented reality environment, only the local representation needs to be changed to be consistent with the user's experience.

A more complex state consistency problem arises when multiple users would like to change the state of a shared environment. These changes can be described as *interactions* instead of just single user actions.

Interaction can be defined as two or more parties performing actions which influences the other. All of the interacting parties need to have the correct VE state in order for the VE to be rendered accurately. Unlike the single user scenario, only changing the local state, will not be sufficient; the local state at the other users will disagree. All the local states of the users need to agree (even though the state is at different terminals) to render the VE correctly. This is known as the distributed state consistency problem.

When all of the local state agree, the agreed upon state can be seen as a universal or *global state*. Similar to [54] we will use *global*

state as the reference or ultimately correct state that all local states should match. Even if there is no single copy of the global state, we will use the term to mean the correct merging/integration of all local states.

In Massively Multiplayer Online Games (MMOGs), an active and related field to MMVE, interactivity is of utmost importance. Interactivity has allowed MMOG games like *CrossFire*, *World of Warcraft*, *Dungeon Fighter Online*, and *The Elder Scrolls Online* to garner great numbers of players: 8 million [55], 7 million [106], 5 million [116], and 3 million [22] active concurrent players respectively. Such large MMOGs need to address the state consistency problem in order to provide players an interactive virtual environment experience worth paying for: the reported lifetime revenue of *CrossFire* is \$10.8 billion and *Dungeon Fighter Online* is \$10 billion [27].

Other than games, the distributed state consistency problem presents itself in many scenarios such as air traffic control, military simulation coordination, distributed database systems and in the future possibly self-driving car swarms.

In distributed database systems, the consistency problem is outlined by Brewers' CAP theorem: A database system can never provide more than two of the three of the following guarantees:

1. Consistency: Every read of the distributed database state will yield the latest written state.
2. Availability: Every read will provide a response immediately.
3. Partition tolerance: the distributed database remains reachable even if network failures occur.

If the network is perfectly reliable and responsive, distributed state can be synced between nodes and consistency and availability can be achieved. However, if a network failure or delay is present, the choice between consistency or availability needs to be made: either deny access to the distributed database until network is restored, thus keeping consistency or; allow access and possibly incur inconsistencies between distributed databases.

In virtual environments these trade-offs also need to be considered. In some cases, such as where authentication or monetary transactions are present, absolute consistency is needed and therefore availability will be restricted. In other cases, such as user movements and non-critical VE changes (eg. interacting with a computer controlled entity), inconsistencies can be tolerated for a small time until the network conditions are restored.

In either case, the user experience of the VE will be reduced: if a feature is unavailable, it might dissuade a user from using the application again. On the other hand, if the VE is inconsistent competition for resources need to be resolved after the network has been restored and state rectification disadvantaging the user could be perceived as unfair.

Therefore we would like to quantify the state consistency and minimise inconsistencies as far as possible; this is called state management.

Users of the VE would like to perform actions to change the VE according to their purposes. For example, in VE games, the purposes could range from benign creative world building such as *Minecraft* or placing traps, walls and barricades for other players in competitive games such as *Fortnite*. In traffic control applications it could be a plane requesting a certain flight path which could influence the other planes.

In the next subsections we will discuss event-based and update-based mechanisms in which state inconsistencies can be resolved and how network conditions could influence them. According to Engelbrecht and Gilmore [54], these are used in typical VE scenarios.

We define the following terms:

- **User:** A User (person) of the VE.
- **Client:** A Client entity representing the user in the VE. Clients receive game updates.
- **Neighbour:** A client entity in close proximity to another client.
- **Host:** A computer capable of running VE application.
- **Node:** A host running the VE software.

2.1.1 Event-based consistency

In event-based consistency, every action taken by a user of the VE is sent to all the other users. Those users can then update their local state to match the changes. However, if different users' action contradict each other, problems could arise. For example, if two players would like to place different coloured blocks in *Minecraft* in the same position or if two planes want to land at the same time, conflicting events need to be handled. In games, this is resolved with game rules and logic: the players that first placed the block gets preference, and the other block is removed. In the aircraft scenario an air traffic controller might give priority to the plane with less fuel reserves.

Event-based consistency mechanisms are common in peer-to-peer network setups. In peer-to-peer networks each node (peer) in the VE has equal authority and uses event messages and game logic to update the local state, thereby achieving state consistency.

Event-based consistency relies on the transmission of events to and from every user of the VE. This might cause a large bandwidth requirement as the event count would grow with $\mathcal{O}(N^2)$, where N is the number of nodes in the network.. This could saturate the network and prevent further events to be sent in time.

Furthermore, if the VE is run on a network with nodes distributed geographically, event messages would not reach their destinations at exactly the same time. Different arrival times could cause conflicts even if game logic is applied: if message 1 arrives before message 2 at node A, but message 2 before message 1 at node B, nodes A and B would process the messages with the same game logic, but the conflict resolution would result in different, inconsistent final states.

2.1.2 Update based consistency

As an alternative to event-based consistency, update consistency gathers all of the user actions at centralised location(s) and there applies game logic to compute a state update. Instead of individual events, the state update is then sent to all of the users of the VE. This drastically reduces the number of messages to be sent to $\mathcal{O}(N)$, where one set of messages are the events messages sent to the centralised location and another set the updates to the nodes.

From this approach it is clear that processing is centralised in order to compute the update message, therefore update based consistency is more commonly associated with server-client architectures. In the next section we will discuss the possible architectures and how they could influence state consistency.

Update based consistency improves upon event-based consistency by allowing a central point of synchronisation: instead of each event arriving at different nodes at an arbitrary time, the sequence of arrivals at the centralised node determines the order in which the events are processed. This prevents conflicting results from the game logic.

However, problems with this consistency mechanism is the requirement of a single (or few) nodes to perform the computation of many other nodes. This strain on computation resources could result in the failure of the server nodes and complete collapse of state consistency.

Unfairness could also be experienced if the time to deliver the events vary between nodes. Events from faster nodes would always be prioritised in game logic computations.

Before we can discuss state management strategies further, we will first need to understand the possible architectures that could support these VEs.

2.2 Architectures

Below is a brief summary (based on [121]) of typical architectures used in Massively Multiplayer Online Games (MMOGs), but the architectures can be used in most VEs.

2.2.1 Server/Client

In the Server/Client (S/C) architecture, clients interact with the virtual world and send changes to the server to be processed. The server processes all incoming messages in sequential order, making changes to its VE state and sends updates to the to each player. If there is a conflict between client actions, the time of arrival and game logic is used to determine which action is performed and which is discarded. The state at the server is described as the authoritative copy. All player changes are evaluated against the server state and incorrect changes are reversed. Yahyavi and Kemme describe this as *Management and controllability* of the VE state, a great benefit for game companies which would like to ensure fairness and generate a profit.

2.2.2 Multiserver/Client

Multiple Servers-Client (MS/C) architectures divide the computational responsibilities among multiple servers. Division of computational load can be implemented in many different ways, for example dividing the VE into different regions or by dividing processing tasks, such as game state updates, in-game messaging, and login authentication. Servers are assigned to the tasks and can be load balanced depending on the computation required. Clients need to connect to multiple servers for full functionality or connections need to be forwarded by a proxy server.

2.2.3 Peer-to-Peer

The alternative to the two above mentioned approaches is a peer-to-peer (P2P) architecture. This means that participants in the VE also perform VE state update computation, messaging, login authentication or even a combination of tasks, similar to what the servers above would have done.

2.2.4 Super-Peer-to-Peer

The above three options represent a broad spectrum of options, but the spectrum could also contain hybrid implementations using elements of both S/C and P2P concepts. We will highlight one such architecture: the super-Peer-to-Peer (super-P2P). In a super-P2P architecture some of the peers are elevated to fulfill responsibilities similar to servers in the MS/C architecture. The other peers can either retain some responsibilities or be demoted to clients. Super-peers are typically users of the network and therefore are not paid for or controlled by a company. The super-peers typically handle tasks that are more difficult to coordinate with many peers. For example, state consistency in a partitioned VE is easier to achieve between a subset of super-peers than between all of the peers in the VE because there are less regions to coordinate.

2.2.5 Considerations for Architecture Design

In the previous section we described four typical architectures.

In the next subsections we will discuss the strengths and weaknesses of each architecture at the hand of Schiele et al's [107] considerations for peer-to-peer MMOGs. In order to generalise, we will only discuss the requirements that fit both P2P and S/C architectures here:

1. Consistency: All of the users should experience the same VE.
2. Persistency: All of the VE state needs to be preserved, even if network or hardware failures should occur.
3. Availability: The VE should be accessible to users constantly without the need to request uptime, especially if the users are paying for the VE as a service.
4. Interactivity: MMOGs and other VE applications typically receive many user events per second. These events should be handled as soon as possible, otherwise the user experience will

suffer. That is, if a user does not receive feedback from an action, she will get frustrated and could leave the VE application.

5. Scalability: MMOGs and other VE applications should support many users (as was seen for *CrossFire* etc. above). Therefore resource utilisation should increase with $\mathcal{O}(N)$ or less and definitely not $\mathcal{O}(N^2)$.
6. Maintainability: MMOGs and VE software need to be updated to repair bugs, prevent cheating exploits and allow game logic changes. The architecture should be able to deal with scheduled downtime to allow updating of VE software.

The architectures are evaluated according to the requirements presented by Schiele et al. above. We also include a subsection about cost.

2.2.6 Consistency

S/C: As there is only one authoritative copy of the VE state, it is kept consistent by the server.

MS/C: If the VE is divided into regions, each server could be responsible for only one region and in this way keep the VE state consistent. Alternatively, servers could keep the VE state of multiple regions, coordinating updates between servers to keep the VE state consistent. The same applies if the functional tasks are divided; coordination is needed.

P2P: Each peer is only responsible for a part of the VE state, and therefore peers will have to connect to multiple peers to see and manipulate the full VE. Latencies between peers could vary widely, making update coordination difficult.

P2P systems are also vulnerable to cheating as it is more difficult to verify distributed state (no easily accessible global state).

Super-P2P: The same challenges are present as with pure P2P architectures. However, the super-peers could be chosen to have better computation and bandwidth performance, reducing the latency to each of the peers in the VE.

2.2.7 Persistency

S/C: Persistence storage could be implemented in S/C architectures by storing the authoritative copy on the server's hard drive. This could require significant hard drive space, so the server would need to be provisioned well.

MS/C: Similar to S/C, the individual servers could store their part of the VE state on their hard drives. Although the same amount of global state still needs to be stored, a distributed storage scheme could be used to reduce the required storage at each server or to add redundancy. Such a storage mechanism has been proposed by Engelbrecht and Gilmore [54], but has not yet been implemented in a real MMVE.

P2P: Similar to MS/C a distributed storage scheme can be used minimising the required storage at each peer. Because there will typically be many more peers in a P2P architecture than servers in an MS/C, the burden of storage space is reduced.

Super-P2P: A distributed storage scheme could be use hard drive space on all peers or only super-peers. Using only super-peers allows for faster retrieval of VE state, but could reduce availability if not all of the super-peers are online.

2.2.8 Availability

S/C: The server needs to be online and active to provide availability. Being online requires the server's Internet connection to be working and not overloaded. Similarly, being active requires the server to be switched on and without fault conditions or overloaded processor. Because both of these rely on single resources (i.e. the single network connection to the server and the single server machine (possibly with multiple CPU cores)), this represents a single point of failure. If this server goes offline or is overloaded, the VE will not be available. This particular problem is that the server is targeted with Denial of Service (DoS) attack, flooding the network connection and making the server unreachable for other users. The S/C could have multiple network cards to prevent network flooding attacks, but the CPUs would still be vulnerable to overloading.

MS/C: Similar to the S/C the servers will need working Internet connections and active, well-provisioned processors. Unlike the S/C, the multiserver approach has more inherent reliability due to the multiplicity of Internet connections and machines. Depending on the network and redundancy designs used in the multiserver architecture, only partial loss (or even no loss) of availability would be experienced by the clients.

P2P: The availability of a resource in VE depends on the availability of the peer responsible for it. In a VE, users are inclined to join and leave at arbitrary intervals. This join/leave operation is referred to as churn. According to [63] a high rate of churn is expected in a P2P network, therefore the architecture designers should con-

sider using replicas of neighbour states to increase redundancy and mitigate the unavailability of some peers.

Super-P2P: Similar to P2P, the availability depends on the individual super-peers. Depending on the super-peer promotion criteria, more peers could be promoted to replace the super-peers that left the VE. However, it could also happen that no peer qualifies for promotion because they have insufficient computational power or bandwidth.

2.2.9 Interactivity

S/C: Interactivity is strongly dependent on the round trip time (RTT) from the client to the server. If a user attempts to change something in the world, the event needs to be sent to the server, processed according to game logic, and the reply sent back to the client. If the RTT is long, the user will experience low interactivity. The RTT is typically related to the geographical distance between the client and server, but could also be influenced by the in-network processing of routers and switches.

MS/C: RTT also influences the interactivity in the MS/C setup, but routing of packets within the server group could add extra transmission and processing time. The incoming event would need to be processed by the correct server in the server group, which could be geographically distant from the client.

P2P: The interactivity is dependent on the RTT to the peer responsible for processing the event. As before, this is also related to the geographical spacing. In some cases this might be beneficial as clients could be located closer together, but could also be detrimental if peers are distant. As clients move around in the VE and change the peer responsible for processing their events, clients could experience high variability in the RTT depending on the processing peers' relative location.

Super-P2P: Interactivity is still dependent on the RTT between peers and super-peers. However, choosing the location of a super-peer geographically in the middle of the connecting peers could even out the latency of the connected peers.

2.2.10 Scalability

S/C: The server needs to have sufficient bandwidth and processing power to compute and transmit all of these updates. Because server hardware capabilities obtainable in a single machine is limited, the scaling opportunities could be enough to support a small number of users, but are ultimately also limited.

MS/C: Multiple servers, even if each is not particularly powerful, allow for a greater total bandwidth and processing power. This allows scaling by adding more servers, but increases the cost by requiring the purchase and maintenance of multiple servers. Load balancing and distributed state management between servers present similar problems as P2P architectures.

P2P: As each peer provides processing power, the computational power grows in accordance with the number of peers in the VE. This allows much greater scaling than S/C architectures, at a fraction of the cost of MS/C architectures. However, as with MS/C architectures, the distributed nature of peers presents a state consistency problem.

Super-P2P: Similar to P2P, super-peers provide processing power. If they qualify, more peers can be promoted to provide more processing power dynamically. Super-P2P provides some of the benefits of an MS/C architecture without the added cost of running more servers.

2.2.11 Maintainability

S/C: If the server needs to be maintained, it will have to be taken off-line. The VE will therefore not be available during this time.

MS/C: Using redundancy mechanisms as explained in the availability section above, individual servers can be taken off-line without affecting availability. This allows the MS/C architecture to provide connectivity to the clients even as maintenance is taking place.

P2P: In order to maintain the P2P architecture, all of the peers need to receive the new software. As the P2P architecture is already established to disseminate VE updates, it should be possible to disseminate game logic software efficiently. However, due to the churn in a P2P architecture, not all of the peers might be online at the same time, and would therefore still be running outdated software. Backward compatibility of the software would need to be considered in order to allow peers to receive updates. In some cases, running outdated software could even allow cheating if the old software had an exploitable flaw. Therefore a software verification mechanism will have to be established in order to guarantee the successful and accurate functioning of the VE.

Super-P2P: All of the challenges of P2P apply. However, as there are fewer super-peers, keeping them updated with the latest software could be simpler than the pure P2P architecture. Once updated, these super-peers can then disseminate the software updates to their connected peers.

Table 2.1: Architecture comparisons based on Schiele et al's [107] requirements.

	S/C	MS/C	P2P	Super-P2P
Consistency	++	+	+	+
Persistency	+	++	+	+
Availability	- -	++	+	+
Interactivity	++	++	+	++
Scalability	- -	++	++	+
Maintainability	- -	++	-	-
Cost	-	- -	++	+

2.2.12 Cost

S/C: A server with high bandwidth and CPU power available will cost a fair amount, but less than the MS/C.

MS/C: The MS/C architecture will cost the hosting company the most of all the discussed architectures. This is due to the multiplication factor applied to an S/C setup costs.

P2P: The P2P architecture will cost the hosting company the least, as users already contribute computing power. However, due to the distributed state problem and possible cheating, a hosting company would likely not be able to charge for a purely P2P system. P2P systems are frequently used by open-source communities. In commercial P2P architectures, the hosting company typically provide centralised services such as authentication which is difficult in a purely P2P system.

Super-P2P: Super-P2P system will have the same minimum cost as P2P systems. The hosting company could possibly provide compensation to the super-peers in order to incentivise becoming a super-peer.

2.2.13 Summary

In summary, state consistency management is easiest when there is only one VE state on an S/C setup. However, S/C architectures are the most vulnerable to failure and could have latencies longer than P2P systems. S/C also does not scale well. MS/C could solve many of the S/C problems, but could be prohibitively expensive to use, especially if large scaling is required. P2P architectures scale the best and in some cases have better latency, but presents problems with state consistency management. Super-P2P architectures solve some of the state consistency challenges by reducing the number of local state copies that need to be synchronised.

Now that we understand the possible architectures, we discuss the scalability, communications and state management strategies available in literature. We will discuss the concepts first and then give a shortlist of example implementations that use these strategies.

2.3 Scalability Strategies

Due to the large number of users that could potentially join the VE, we need to have a system that can handle all of the processing and network traffic. In this section we will discuss Area-of-Interest management which allows us to limit the update dissemination required, world partitioning and distributed hash tables, both of which allow us to distribute the state and processing load.

2.3.1 Interest Management

In typical VE (and VR) applications, the users have a limited sight radius (only part of the VE state is visible to the user) and limited interaction range (only objects/environment within reach can be manipulated). This means that each user is only interested in an area around her, defined as an Area-of-Interest (AOI). Only events / updates to that region will make a perceivable difference to the user. By only sending relevant events / updates to the user, the communication requirements can be relieved. Reducing communication requirements allows the network to support more users.

Similarly, the user could only be interested in some of the messages on the server (for example private group messages) and therefore do not need to be informed of the global messaging. Keeping track of users' AOIs and other interests, and sending them only relevant updates is called interest management.

Interest management can be applied in all of the above mentioned VE architectures.

2.3.2 Partitioning Schemes

In order to divide the processing tasks in MS/C or P2P architectures, some distribution scheme is needed. Task distribution can happen on various levels such as:

- **Role:** Each server or peer handles a role with tasks such as managing in-game chat, computing or distributing updates, computing user interest, filtering updates and storing VE state for redundancy.

- **Spatial Partitioning:** Each server or peer is assigned a region in the VE to control. They are responsible for managing state consistency, computing and distributing updates in that region. Redundancy can also be achieved for example by making multiple servers / peers responsible for each region or by allowing servers/peers to be backups for adjacent regions.

By dividing the tasks in one of these ways, more servers/peers can support the computation required in the VE, thus increasing the capabilities of the VE system and allowing more users to join the VE. However, the distributed state consistency problem needs to be addressed, especially if the VE is divided into regions, as the users will likely move between regions and should always experience a consistent VE.

2.3.3 Distributed Hash Tables

Distributed Hash Tables (DHTs) allow data to be stored by using hash keys (hashed according to a property of the data) among peers. DHTs allow the state of objects, entities and regions to be distributed between peers in an easily retrievable way. The required state of an object can be retrieved by generating the hash key and querying the DHT. The query is forwarded between peers until the peer responsible for storing the object is found. This peer then replies to the original query. DHTs can be randomly distributed, optimised to store data in a network locality (peers in the same network are also responsible for nearby parts of the DHT key space) or optimised to store data in VE locality (peers in close VE proximity will be responsible for nearby parts of the DHT key space).

DHT enables the fair distribution of state of objects, entities and regions between peers (or even multiple servers). This means that the load is balanced between all the peers or servers, allowing each to accept the maximum number of users. In the case of MS/C, if more processing power is required, more servers can be added and the processing load would be evenly distributed to the new servers.

The main disadvantage of DHTs is the latency associated with completing queries on the DHT: because the query needs to traverse the DHT until it reaches the peer responsible for storing the data (which could take multiple hops depending on the DHT used), the query could take time to complete. In cases where responsiveness is of utmost importance (like First Person Shooter games [90]), this latency could be unacceptable.

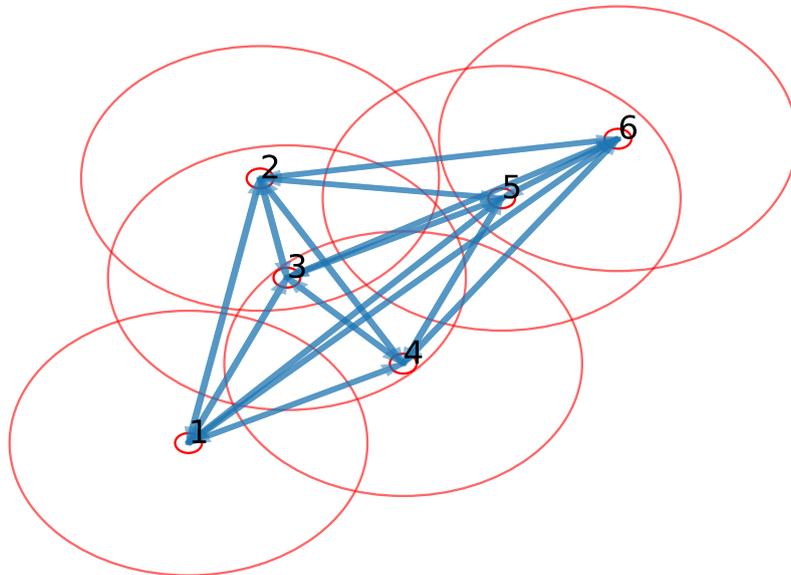


Figure 2.1: Direct Connection between peers. The inner red circle represents the client and the outer circle the client's AOI. Blue arrows indicate connections.

2.4 Communication Strategies

VEs need to communicate many different messages including VE chatting, transactions and VE state updates. Messages have different levels of acceptable latency: a VE state update might be needed as soon as possible, whereas a chat message or in-game transaction might be acceptable even if it takes a while to reach its destination, possibly with higher reliability. In this section we will discuss typical communication strategies and their advantages and drawbacks.

2.4.1 Direct Connection

The simplest communication scheme is direct connections between all clients in the VE (shown in Figure 2.1). In P2P architectures this would allow the direct dissemination of events with minimum latency, leading to a high state consistency overall.

However, as was mentioned above, the bandwidth requirements and numbers of active connections in the direct connection strategy

would grow with $\mathcal{O}(N^2)$ and would therefore be unsustainable for large numbers of users.

There are three approaches to address the $\mathcal{O}(N^2)$ problem: Compute and disseminate updates, use forwarding instead of direct connections and use interest management. Interest management was discussed in Section 2.3.1, and forwarding will be discussed in the next subsection.

By computing the changes to the global state and sending updates instead of events the required bandwidth is reduced. Computed updates are a summary of all the events that changed the global state, and would therefore require less bandwidth to transmit. It is not always possible to compute changes to the global state as the full state is not available, but changes to the local state can be computed and sent.

In order to further reduce the required bandwidth, *delta encoding* [35] can be used. Delta encoding means that only the difference in state is transmitted. For example, in a player versus player combat scenario, instead of sending the entire state of a player each time it moves, only a position difference vector and difference in health can be sent. This reduces the number of bytes in each update message and therefore also the bandwidth required. However, if an update message is lost (possibly due to packet loss), the lost difference vector would not be applied, leading to the local state drifting from the global state. By sending absolute updates (i.e. full state) periodically, the local state can be corrected. Delta encoding is not only limited to the direct connection strategy, but is applicable to all of the following communication strategies as well.

2.4.2 Forwarding

In order to reduce the number of active connections, the messages are transmitted using a forwarding scheme. Instead of sending the messages directly, the messages are sent to the neighbour nearest to the destination node [78]. An example forwarding scheme is shown in Figure 2.2.

This could mean that multiple hops may be required to complete the transmission, eg. if node 1 should want to send a message to node 6. In order to ensure that all nodes are reachable, nodes have to retain connections to their neighbours. Furthermore, simple forwarding could mean that multiple copies arrive at the end nodes if there are multiple paths between nodes in the network, wasting bandwidth.

Spanning trees can be used to calculate optimal forwarding paths, preventing redundant packet sending. Spanning trees connects all

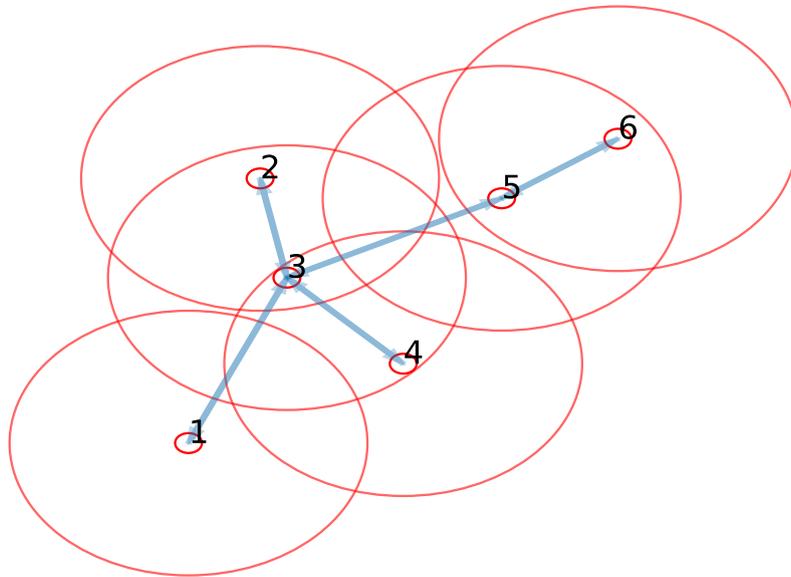


Figure 2.2: Forwarding among neighbours.

of the endpoints while minimising the number of connections in the network. The first step in constructing a spanning tree is choosing the root node, typically the server or a peer sending a message. The other nodes then determine the path cost of each route to the root node, based on hop count, latency or VE distance, depending on the implementation. Each node selects its parent node with the lowest path cost to the root.

Figure 2.3 shows a spanning tree forwarding scheme with path cost based on the VE distance. The big purple arrows show the unique paths from the sending node to all of the neighbours interested in the message. The smaller purple arrows indicate redundant paths which are not selected in the spanning tree as they have higher path costs. In some implementations, redundant paths are also kept in order to assist in path recovery in case a forwarding link fails, for example the Rapid Spanning Tree Protocol [14].

Spanning tree forwarding reduces the overall bandwidth use and latency as messages are only forwarded along the ideal path. Compression can also be used on the message traveling along the spanning tree which could further reduce the required bandwidth [75].

Spanning trees are constructed using lowest latency, hop count

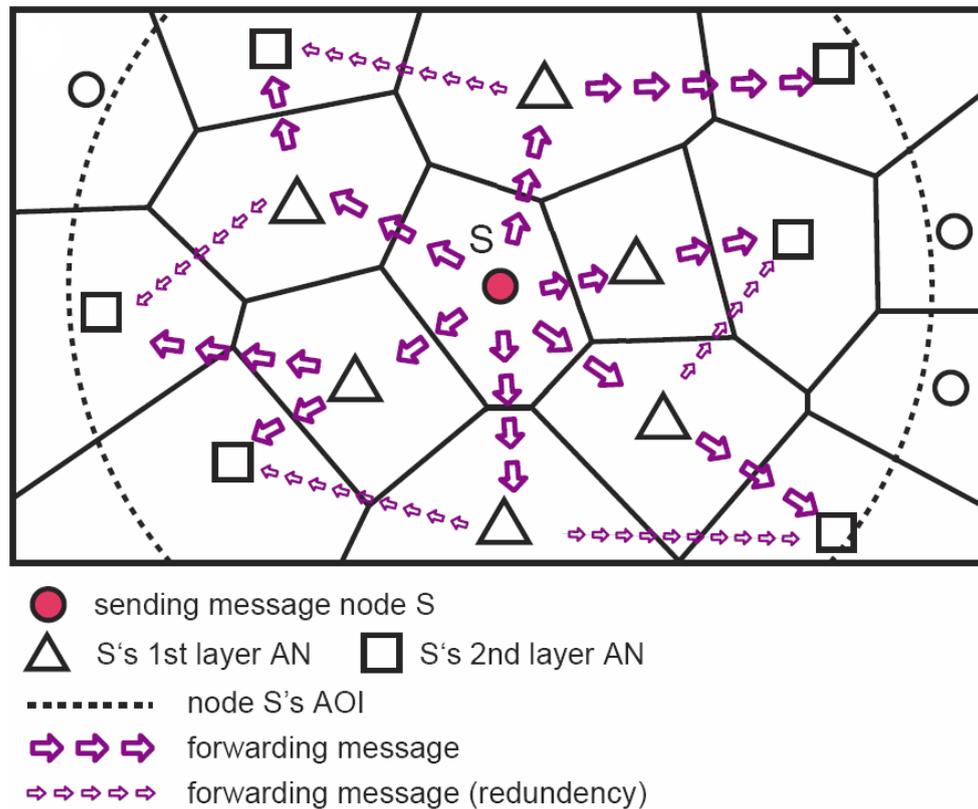


Figure 2.3: Spanning tree forwarding. Source: [44]

or VE distance between nodes as a criteria for efficient forwarding. Constructing the spanning tree therefore requires accurate knowledge of either these properties. In a dynamically changing environment, determining these properties could introduce a large overhead, for example, multiple ping messages each update step to determine latency.

Relaxing the criteria and allowing unoptimised spanning trees could reduce the overhead at the cost of extra bandwidth or transmission delay.

One of the caveats of forwarding and spanning trees is the sensitivity to packet loss: if a packet should be lost on a forwarding path, all nodes that receive forwarded packets will also experience packet loss.

To summarise: forwarding can reduce the number of connections that each node needs to maintain. In order to make forwarding efficient, optimised spanning trees can be used, but could require extra overhead to calculate. Note that in both simple forwarding and spanning tree forwarding, transmitting over multiple hops could in-

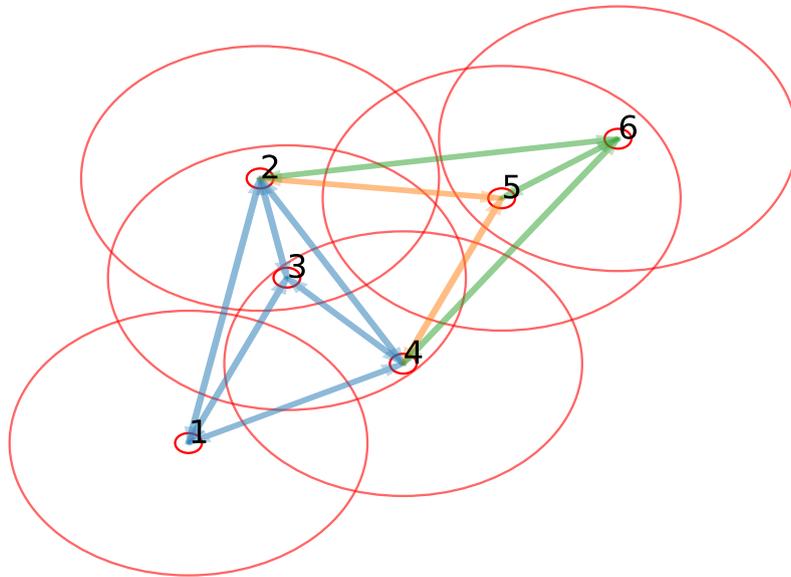


Figure 2.4: Multicast groups.

introduce excessive transmission delays for nodes far away from the root node. Furthermore, if a transmission should be lost on a forwarding link, the packet would be lost to all further nodes on the forwarding tree.

2.4.3 Multicast Groups

Multicast groups can be established using IP multicast [88] or application layer multicast [48].

IP multicasting allows the distribution of messages to group members without requiring the sender to send multiple copies. The intermediate routers are responsible for copying the incoming multicast messages to the interested receivers. Interested receivers communicate their interest in the multicast by sending a multicast group join message to the router. Routers then use this list to send the incoming message to members of the multicast group.

However, IP multicasting is not supported by all routers on the Internet [48]. In order to generalise multicasting to support all network hardware, Application Layer Multicast (ALM) was proposed [48]. ALM maintains an overlay which retains connectivity with all the con-

nected nodes using unicast. Using the ALM overlay, nodes signal their intention to join a multicast group. This intention is recorded by the overlay and all messages addressed to the multicast group is forwarded along the overlay to the group members.

Multicast groups can be established for updates concerning a specific object, entity or region in the VE. Figure 2.4 shows such multicast groups in different colours. Using IP multicast, the total bandwidth can be reduced as the message is only copied to the group members at each router. Using ALM abstracts the communications from the application allowing overlay optimisations without changing the application. ALM also uses spanning trees to improve bandwidth use.

2.4.4 Spatial Publish Subscribe

Spatial Publish Subscribe (SPS) [68] implements interest management on a communication level. A location-specific event is referred to as a spatial publication. Clients can listen for events near them by subscribing to the region that they are interested in. The SPS communication layer then sends the spatial publications that fall within the subscription regions to the respective clients. The advantage of SPS is that it limits the number of messages that need to be sent in each update step by communicating only the necessary events to the interested clients.

Typically clients subscribe to their AOI region and interact with entities and the VE in their AOI, thereby publishing events also in their region. Figure 2.5 shows the communication paths of clients that fall within each other's AOIs. As clients move in the VE, their AOI will change and therefore they need to change their subscribed region accordingly.

SPS allows the efficient transmission of events to interested peers, but does not provide connectivity for the entire network. Therefore another communication strategy is required to support the SPS communication layer in administrating the subscriptions of all nodes. Thus, SPS adds a small overhead for transmitting subscriptions, but the overall bandwidth reduction justifies the overhead.

In summary, each of the communication strategies has advantages and drawbacks. It is up to the VE designers to make a trade-off between communications strategies or to combine strategies in order to achieve the advantages of both.

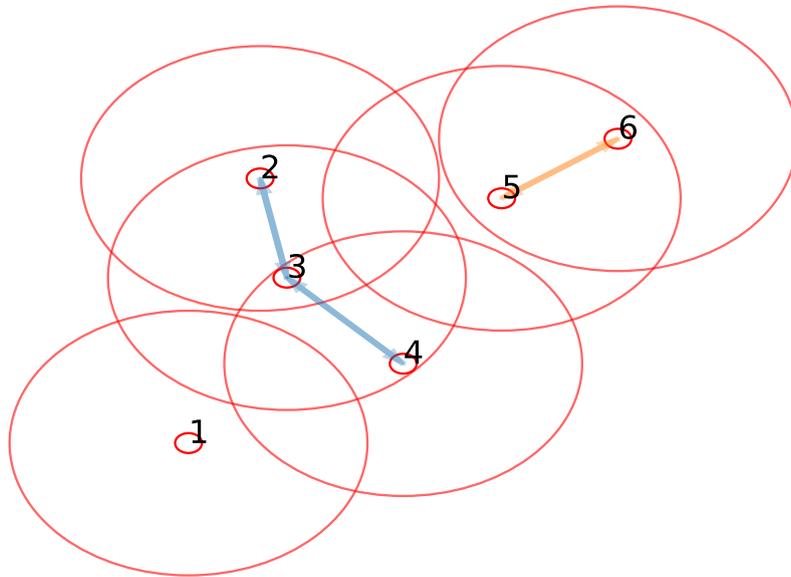


Figure 2.5: Spatial publish subscribe.

2.5 State Management Strategies

In this section we will discuss strategies for improving state consistency by using prediction and synchronisation strategies.

2.5.1 Dead-reckoning

Dead reckoning is a way of predicting the future position of an entity or player based on their current position and movement vector. In VEs, this strategy can be used to hide temporary connection loss by moving entities *in the direction they were traveling*. This would give the impression of an interactive VE, although the clients are not receiving updates. However, once connectivity is restored, it could be necessary to correct the state if the entity has changed its movement vector during the disconnected time. This correction of state would happen suddenly and would be experienced as an abrupt movement in the VE, possibly confusing the users. Dead-reckoning can also be used continually, with the state updates only correcting errors by sending the residuals of subtracting the motion prediction from the global state. This process can be compared to MPEG's motion

compensation coding [58].

2.5.2 Bucket Synchronisation

Bucket Synchronisation, originally presented by Diot and Gautier [51], is a discretisation process typically used to reduce the detrimental impact of varying latencies between clients and the server or between peers. It is implemented by adding a small lag time before events are processed, gathering all events and processing the events in a single batch. Updates are also sent in batches, allowing clients with worse connectivity to receive updates at a similar rate and delay as the clients with faster connections, increasing fairness. In addition to providing fairness, the latency compensation of bucket synchronisation improves state consistency between global and local states by controlling when updates occur, synchronising client local states updates.

Bucket synchronisation also fits with typical game designs where game logic is applied in update steps, at rates of 10-20 times per second [121].

2.5.3 Time Warp

Jefferson [74] presents an algorithm for synchronising events with virtual time, a time dimension unconnected to normal time. Jefferson proposes that if causality is maintained in virtual time, distributed computation processes can be synchronised irrespective of the real-world computational time that is required to complete their tasks. Synchronisation is achieved by timestamping each interaction between processes with a virtual send time and virtual receive time. The virtual send time is determined by the local process clock. The virtual receive time is the specified time that the message should be received. If the receiving process clock has not reached the virtual receive time, the message waits in a buffer until it has. Depending on the realtime it takes to complete its computational task, a local process clock proceeds faster or slower than real time, thus warping time. If an inconsistency in distributed state appears, for example because a process result was not taken into account in a following computation, the process can be rolled back to a previously correct virtual time and be recalculated using the correct state.

Similar to bucket-synchronisation, time warp can be used to reduce the effect of varying latency. By timestamping events at the different nodes, the virtual time of the independent processes can be synchronised and handled by the server or super-peer in correct order, despite the latency of transmission.

In summary, dead-reckoning, bucket synchronisation and time-warping attempt to compensate for network conditions by predicting or restricting updates in the application. Application based state consistency strategies are limited in that they can only use domain knowledge to mask state inconsistency and are unable to compensate for truly adverse network conditions.

2.6 Related research: VE implementations

In this section we will discuss example VE implementations that use the above-mentioned scaling, communication and state consistency strategies. At the end of the section we will compare the implementations. In the next section we will discuss why we chose VAST [70] as our testbed implementation.

2.6.1 COVER

COVER [92] is a super-P2P system that uses quadtrees to divide the VE into rectangular regions. Quadtrees are a recursive region division algorithm where each rectangle can be divided into four quadrants, each which can be divide into quadrants, etc. Each region is assigned a super-peer: the peer closest to the center of the region is promoted to a super-peer. In terms of communication, COVER uses a direct connection inside peer AOI and a forwarding approach to other peers. Nodes send their movement updates to all neighbours in their AOI as well as the super-peer responsible for the region. AOI neighbours forward the message to their neighbours and so on. Peers that do not have AOI neighbours receive updates from the super-peer of the region. The advantages of this system include scalability, as processing and message dissemination is divided among peers and super-peers. The drawbacks of COVER include processing overhead of establishing quadtree regions with moving peers, and high bandwidth use in if there are many neighbours inside their AOI.

2.6.2 Colyseus

Colyseus [36] is a P2P system which uses its a range queriable DHT for storing objects in the VE. The storage is implemented with Mercury [37] which allows the storage of multi-attribute variables. Mercury uses a DHT for each attribute of the stored values, eg. x-coordinate, y-coordinate, player name, etc, referred to as dimensions of the data. Each value (block of data containing all the attributes)

is duplicated in each of the DHTs. In order to perform a look-up, any of the dimensions can be queried to obtain the value or values required. The DHTs are stored on the peers; peers could even host parts of multiple DHTs. As data is not distributed according to normal cryptographic hash functions, load balancing is needed to ensure that the peers host equal parts of the DHTs. The size of the key-space responsibility at each peer is different as the load balancing attempts to distribute the values evenly across the available peers.

For example, peer 1 would be responsible for x-coordinates 0-60 with values at 23 and 48, while peer 2 is responsible for x-coordinates 61-180 with values at 71 and 140. Therefore both peers have the same number of values stored, but different key-space sizes.

In Colyseus the owner of an object (the peer responsible for storing it) is also responsible for receiving events, calculating and sending updates about it. Other peers that are interested in the object subscribe to it based on an attribute (eg. x-coordinate changes) and the owner periodically publishes state updates to those subscribers.

Colyseus subscribes to the objects in the user's AOI as well as using dead-reckoning to subscribe to objects that could come into the AOI as the user moves. Colyseus also pre-fetches the required objects. That is, it retrieved and stored in the local state before it is necessary (based on the dead-reckoning prediction) reducing the experienced latency from querying the DHT. However, due to the dynamic nature of a VE, there is a processing overhead of continually moving the objects in the DHT as well as load-balancing the DHT as stored values move in the key-space.

Despite this, Colyseus has proven that it can support "an order of magnitude" more players in *Quake 2* (a latency-sensitive first-person shooter game) than a similar S/C architecture setup.

2.6.3 Apolo

Apolo [82] is a P2P system that uses forwarding to disseminate events. In order to reduce the bandwidth required and the number of connections to maintain, each peer only sends events to four others, specifically chosen to be the closest peers (in terms of the VE) in four different directions (see Figure 2.6). A coincidental benefit of sending events to the closest peers is that some of the AOI neighbours would be the first hop peers.

However, limiting the number of connections increases the latency of message dissemination because multiple hops are required for messages to reach their intended destinations. Allowing only four

Motion smoothing filters are applied to eliminate jerky and unrealistic motions. Updates to these objects are called guidance and the predicted movement model smoothly corrects the inconsistencies by incorporating the guidance in its movement model.

In terms of communication, Donnybrook uses multicast trees. At each update step, each peer generates a multicast tree from the source to interested users by selecting paths via neighbours to deliver the updates. These paths are not optimised to be the shortest, as that would require coordination between nodes, which could incur latency and computation costs. In order to ensure the messages are delivered and reduce the total bandwidth required of one peer, a forwarding pool is made available to the network. Peers in the forwarding pool have spare bandwidth and can forward messages in addition to their own responsibilities. Peers that require extra bandwidth can assign peers from the forwarding pool to send their packets along the generated multicast trees.

The creators of Donnybrook concluded that the system can support an estimated 900 players in a Quake 3 game within acceptable latency standards. However, the author concedes that further experimentation on large scale needs to be conducted to prove the scalability. Furthermore, it is unclear if interest sets can introduce more opportunities for cheating and if an interest size of five is valid for large scale VEs.

2.6.5 Badumna

Badumna [80] is a super-P2P system that divides the VE into a static grid of rectangular regions as well as dynamic bounded regions in areas of high interactivity. The regions are assigned to super-peers using a DHT.

The super-peers for cells and regions query the DHT every update interval for the state of the entities within their region, and distribute updates to the peers in their region. Peers within a region are also allowed to communicate the updates amongst each other (referred to as a “gossip” protocol) in a fully connected manner to reduce the network load on the super-peer.

The drawbacks of Badumna include the computational cost of managing dynamic regions (especially with moving peers in the VE) and latency introduced by all the queries on the DHT for state, especially if the required data is further away (in term of key space) from the super-peers that request it.

2.6.6 Voronoi Self-Organising

Voronoi Self-organising (VSO) [70] is a super-P2P system which uses Voronoi diagrams to partition the world with SPS and forwarding to disseminate messages. Each partition in the world is assigned a super-peer called a (interest) Matcher. The Matcher is responsible for gathering all the publications to its region and disseminating them to the correct subscribers, thus matching the interests of the publishers and subscribers in the region.

As peers move around in the VE, their AOIs and therefore their subscription areas will move. If the peer moves across a matcher's region boundary, the subscription needs to be moved to the new matcher. VSO defines an explicit ownership transfer procedure which ensures that the new matcher acknowledges the transfer before the subscription is deleted at the original matcher. Each peer connects directly to the owner matcher of their subscription.

The matchers use the Voronoi Overlay Network (VON) [69] to organise their responsibility regions. VON (using Voronoi diagrams) divides the VE into regions such that each point in the region is closer to the region's matcher coordinate than any other matcher. VON requires that each matcher keeps a list of enclosing neighbours and connect to them. Enclosing neighbours are the matchers that surround the region the current matcher is responsible for. By using the enclosing neighbours as forwarding paths, all matchers can communicate to each other and keep the world partitioning consistent. The forwarding latency between matchers are acceptable because the region's boundaries do not change quickly and are allowed to be eventually consistent.

Finally, VSO provides load balancing by dynamically decreasing the size of regions of a matcher or by promoting a peer to be a new matcher. In order to achieve this in the Voronoi diagram, other matchers either need to be moved closer to the overloaded matcher or a new matcher can be placed such that it divides the overloaded matcher's region. As the regions change, some subscriptions are transferred to neighbouring matchers, reducing the load.

The downside of VSO include the cost of recalculating Voronoi diagrams as load balancing occurs and the requirement of matchers to maintain connections to all of its enclosing neighbours. However, VSO provides scalability with dynamic regions and matcher promoting, as well as reliability in terms of subscription transfer. Normal peers in the network require the minimum network bandwidth as they only receive events that they are interested in.

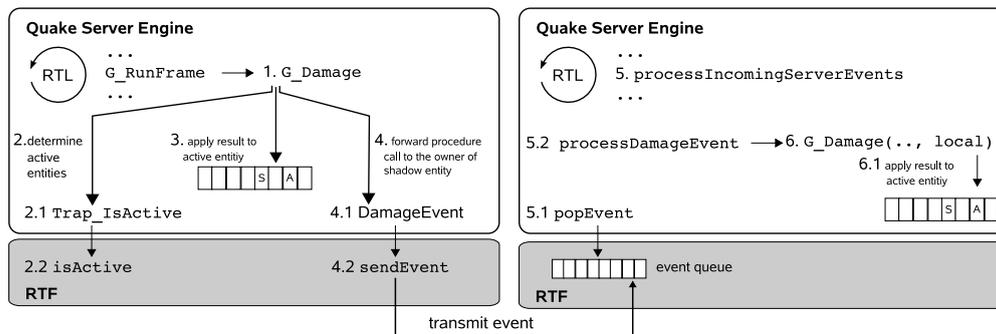


Figure 2.7: Real-Time Framework (RTF) interactions. Source: [100].

2.6.7 Proxy-network topology

The proxy-network topology [93] is an MS/C system where game state is replicated on each of the multiple servers. The responsibility for calculating state updates of entities are distributed among servers. In this system *active* entities refer to those that need to be computed on the current server and *shadow* entities are computed on other servers. Each server computes the updates to their active entities and then disseminates these updates to the other servers to update their shadow entities.

Clients can join any of the servers and will receive updates relevant to its AOI from the server's replicated state. In order to reduce the latency of shadow entities, the A* path search is used for both active and shadow entities. Waypoints of the path search algorithm, instead of actual movement coordinates are sent over the network in order to reduce the required bandwidth.

It was demonstrated that around 290 active users can be supported in Rokkatan, the real-time strategy VE created to test the proxy-network setup. The benefits of this system include lower bandwidth use for inter-server communication and scalability by using multiple servers. The drawback of this system is that the state of the entire VE needs to be stored on each server.

2.6.8 Real-Time Framework (RTF)

Real-Time Framework (RTF) [100] is middle-ware that allows the use of a multiple servers to support clients. Middle-ware is an independent communications layer that allows applications to be distributed with the minimal changes to server and client software. The creators of RTF demonstrated the effectiveness of their system by implementing it into a Quake 3 (first-person shooter) game. Similar to the proxy-network topology, they use state replication and distributed

computation. However, they did not use movement prediction, but rather sent precise movement updates from the clients to the RTF middleware. The RTF middleware then determined which server is responsible for processing the event. If the event involved entities from multiple servers, RTF requested the required state from the servers in order to complete the computation. Figure 2.7 shows a typical interaction when a rocket is damaging a player entity stored on a different server.

Using RTF, the modified Quake 3 could support 85 clients simultaneously on four servers compared to the 32 clients with the single server. All latencies were under 50 ms, which was deemed acceptable for fast-paced first-person shooter games. However, it was observed that the extra inter-server communication used more processing and resulted in extra latency such that each server could only handle 21 clients in the four server setup, compared to 32 clients on the single server setup.

2.6.9 Summary of related research

Table 2.2 shows the advantages and disadvantages of each of the discussed VE implementations. As can be seen, there is a trade-off to be made: fewer connections versus lower latency, more efficient storage lookup versus less computation overhead, etc. Each VE designer should weigh the option in order to best suit the needs of the VE.

In the next subsection we will discuss the VE implementation that we chose to use in our evaluation. We decided to use an existing VE implementation that already addresses the problems of scaling and state consistency, so that we could focus on the effect of loss on the VE state.

2.7 Why Spatial Publish/Subscribe and Voronoi Self-Organising?

In this dissertation we want to study the effect of adverse network conditions on the state consistency of a VE.

In order to quantify this we need the following four properties of VE implementation:

1. Measure state consistency on the VE level. This will give us an accurate description of the experience that users of the VE will have.

Table 2.2: Summary of VE implementations

Implementation	Advantages	Disadvantages
COVER [92]	Message forwarding outside AOI	Quadtree computation, high bandwidth in crowding.
Colyseus [36]	Distributed storage, pre-fetching	DHT reordering and load-balancing
Apolo [82]	Multicast trees (lower bandwidth)	Multi-hop latency
Donnybrook [35]	Interest sets, predicted movement model and multicast trees reduce bandwidth	Possible cheating, possibly less immersive and interactive.
Badumna [80]	Distributed storage with DHT, dynamic regions allow efficient scaling when crowding occurs	Overhead of computing dynamic regions, latency introduced by DHT queries.
VSO [70]	SPS allows efficient communication, dynamic regions allow scaling	Overhead of calculating VON regions
Proxy-network [93]	Shadow entities reduce computation, path prediction reduces bandwidth	Full VE state stored at each server
RTF [100]	Shadow entities reduce computation, middleware require minimum changes to server and client	Processing and latency because of frequent inter-server communication

2. The VE implementation should *not* include network failure compensation mechanisms so that state consistency can be accurately correlated with the applied adverse condition.
3. The VE implementation should be open source so that we can extend the code where needed, for example to implement TCP or UDP communications.
4. The VE implementation should already support interest management and scaling so that results gathered in this study can be relevant to the rest of the research field.

In order to satisfy these requirements, we chose the VSO implementation. It already provides state consistency metrics built into the code. It does not provide network failure compensation and the SPS nature will make the state consistency very dependent on network state, thereby enhancing the measurability of adverse network conditions. The C++ implementation is open source under the LGPL license so it can be extended with new functionality. Finally, VSO supports scaling and interest management with peers being promoted to matchers as needed.

2.8 VAST Library

VSO uses the VAST¹ library [23] which implements all of the software components to represent matchers and clients, and calculate Voronoi diagrams.

Hu's [69] paper introduced the Voronoi Overlay Network (VON), which uses Voronoi diagrams to partition a virtual environment into regions and associate each region with a peer. A Voronoi diagram partitions an area (in our case a Virtual Environment (VE)) into regions such that every point in a region is closer to its peer than any other peer. Neighbours are peers that are in adjacent regions to the peer. Direct connections are made to all Area-of-Influence (AOI) neighbours ensuring low latency interactions with the peer. Peers also keep direct connections with enclosing neighbours (all surrounding neighbours) even if these fall outside of the peer AOI. This prevents overlay partitioning (division of the overlay into unconnected groups) as all peers can connect with other peers through their enclosing neighbours.

In [71], Hu et al. extend VON by introducing Spatial Publish/-Subscribe (SPS). They identify spatial (i.e. VE location specific) area

¹VON-based Application-layer Spatial publish/subscribe Topology awareness (VAST)

or point events as the basic state update mechanic in virtual environments and explain that all VE interactions can be described as an SPS action.

In general SPS functions as follows:

1. Any entity in the VE can subscribe to events in an area in the VE, typically called their Area-of-Interest (AOI). Multiple subscriptions per entity is allowed. Point subscriptions are also allowed, but are defined as an area with zero size. Therefore all subscriptions are treated as areas.
2. Any entity can publish an update to any area in the VE.
3. At the points or areas where subscriptions and publications overlap, published events are sent to the subscribers. This process is referred to as interest matching.

In a pure P2P system, peers signal their interest by sending a subscription message to publishers interacting with their AOI. Publishers keep a list of subscribers and forward events of the area to all of the subscribers.

However, in order to reduce the multiple transmission required for P2P SPS, Hu proposes S-VON, a spatial publish/subscribe extension to VON. S-VON promotes hardware capable peers to super-peers to implement relays in the network. Instead of sending spatial events to all of the subscribers, the publishers simply send the events to their closest relay. In S-VON, super-peer relays keep a list of publisher/subscriber mappings and forward events to the subscribers accordingly. This reduces the bandwidth requirements of the publishers as the relays perform the forwarding, instead of the publishers sending multiple unicasts.

Note that the client location in the VE can be different from its connections in the physical world. Figure 2.8 shows the mapping between virtual environment entities and S-VON overlay.

Finally, in [70], Hu and Chen adds Voronoi Self-organising (VSO) to S-VON, as was described in Section 2.6.6. VSO enables load-balancing of relays (in this paper called Interest Matchers) by partitioning the VE and assigning partitions to each matcher. If a matcher is overloaded, regions can be repartitioned or peers can be promoted to super-peers to deal with the increase communication load.

There are thus two roles in the VAST environment: *clients* are normal peers without any computational or forwarding responsibilities and *relays/matchers* assist in forwarding events between publishers and subscribers.

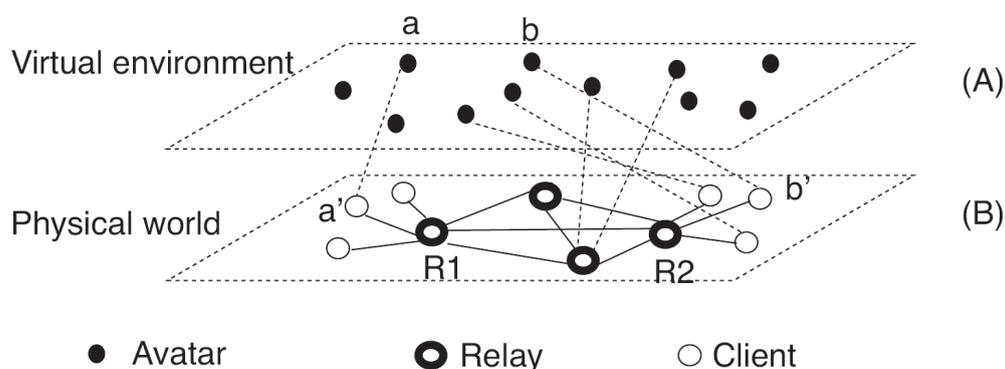


Figure 2.8: S-VON architecture, relays represent super-peers with connected clients, *Source: [71]*

All of the work described above is implemented in the latest version of the VAST library which will be used for our evaluations.

SPS is primarily a communication layer. It allows efficient, interest matched communication to keep the VE state intact by disseminating events. However, if complex game logic needs to be applied to the VE, state managers need to be added to the system. For example, when a player character needs to be a certain level in order to interact with an entity, the game state manager can apply game logic to the interaction event and refuse it if the player character does not qualify.

Although Hu does not describe the role in detail, they mention that state managers could be integrated with SPS. According to Hu, state managers can integrate with SPS system by subscribing to spatial events, applying game logic, calculating state updates and publishing the updates to the SPS, which will then send the updates to the correct subscribers.

2.9 VAST as a Case Study

VAST fits our requirements of a VE implementation and related work provides reference results for comparison to our study. However, no single VE implementation would suit the needs of every MMVE, VR or AR application. We therefore use VAST as a case study for the effect of adverse network conditions on state consistency. In the concluding chapter we will attempt to place our results of VAST in the context of general VE applications as well as explain the limitations of our results.

2.10 Application Domain

The envisioned application in this dissertation is a VR, AR or MMVE experience shared by users in close proximity, with all user devices connected to the same switch or router. The advantages of this application includes minimum network delay as well as many network coding opportunities. This application domain is especially relevant for AR applications where the environment is locally specific and therefore the users would also benefit from proximity based communications. In literature, [77] show that this type of network is well suited to network coding.

Although this application domain limits the generalisability of the results, we propose that the methodology used in this project can be used for larger and more complex networks. Valuable future work would include comparing results of a larger network to results obtained in this project.

2.11 Summary

In this chapter we discussed the distributed state consistency problem and the various proposals to reduce state inconsistency. We discussed the advantages and disadvantage of Server/Client, Multiserver/Client, Peer-to-peer and Super-Peer-to-Peer architectures in the light of Schiele's [107] architecture considerations. We summarised typical scalability, communication and state management strategies found in literature as well as discussing specific implementations. Finally, we discussed the VAST library (the combination of VON, S-VON and VSO work) in detail and explained why we will be using it in our evaluations.

Chapter 3

Adverse Network Conditions on State Consistency

In this chapter we will discuss the adverse network conditions experience in wireless networks. We will also discuss how these network conditions could affect the state consistency of a networked virtual environment.

3.1 Adverse Network Effects Definition

3.1.1 Packet Loss

Packet loss is defined as the failed transmission of a packet from the sender to the intended receiver. Depending on the transmission medium, this failure could be partial (partial or corrupted packets can still be received) or completely lost. Typically verification mechanisms are used to determine if the packet was received exactly as the sender intended. If the packet was received partially, the receiving side could decide to use the partial packet, or discard it completely. In the latter case, users of the receiving side hardware would experience the packet loss as complete due to the abstraction of the receiving side hardware.

Although the explanation why wireless networking is generally a lossy physical layer is beyond the scope of our work, we present three papers which explain possible reasons. They explain the concepts based on Wi-Fi as the networking protocol, but the problems are similar for most wireless networks.

Firstly, Wi-Fi operates in the Industrial, Scientific and Medical (ISM) band. Although transmission power is limited in this band, devices are subject to interferences from many devices including Bluetooth, Zigbee and other wireless radios as well as general interfering

devices such as microwave ovens. Zeb, Asim, Ali and Mufti [123] investigate the effect of interference of two microwave ovens on the throughput and reliability of Wi-Fi stream between an access point and receiver. They found a significant increase in transmission errors, indicated by failed CRC checks of the Frame Check Sequence field.

Another possible reason for loss in Wi-Fi is collisions due to the hidden node problem. Wi-Fi uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) which detects if a carrier is present before sending. If the wireless medium is free, a Request-To-Send (RTS) is sent to the access point. The access point replies with a Clear-To-Send (CTS) if the medium is free and to let other connected Wi-Fi clients know to wait before transmitting. However, due to the physical placement of Wi-Fi access points and clients, an RTS or CTS may not be received. This is referred to as the hidden node problem. Such a hidden node could start transmitting while another node (who sent the RTS) is transmitting. This causes a collision and the data packet could be lost.

Biswas et al. [38] examined a subset of Wi-Fi router measurement records of the Cisco Meraki platform. Cisco Meraki gathers data from many access points worldwide. Biswas studied a subset of the entire dataset consisting of records from 10000 devices. From this dataset they determined that each Wi-Fi access point has an average of 55 nearby networks, i.e. access points in the same area. These nearby networks cause interference, affecting packet delivery. Cisco Meraki periodically broadcasts a 60-byte packet to determine link quality to Meraki routers around it. These broadcast packets, especially in the 2.4 GHz band, show a low delivery ratio. If the Wi-Fi channel is very busy, the access point cannot get a transmission opportunity for the IP broadcast. The low delivery ratio signifies that the Wi-Fi channel is continually busy. Although IP broadcasts are not the typical use of the Wi-Fi channel (priority is given to unicast), the continually busy channel can cause unicast transmissions to fail more frequently and contribute to the hidden node problem.

Finally, Simic, Riihijarvi and Mahonen [112] investigate the effect of 802.11ac Wi-Fi 80 MHz wide channels in indoor environments. The Wi-Fi 802.11ac standard was introduced in December 2013, and is currently the latest Wi-Fi standard which is commonly in use. Simic showed that 80 MHz channels can significantly improve bandwidth in sparse environments, but Adjacent Channel Interference (ACI) cause throughput to drop in a dense network environment, that is many access points and clients. This is because wide channels interfere with nearby channels and in the worst case experience interferences from multiple APs at once. In Simic's exper-

iments they identified the "frequency flow-in-the-middle" problem: an AP and client that function in a channel located between APs in higher and lower frequency channels experience ACI from both sides. The constructive interference cause the collision avoidance algorithm to place the AP continually in the back-off state. They describe this AP as "starved" of transmission opportunities, resulting in loss and / or many retransmissions.

These articles show that loss is typically present in Wi-Fi networks and we therefore investigate the effect of loss on an interactive application such as a networked MMVE.

3.1.2 Network Delay and Network Delay Variation

Network delay (or latency) is defined as the time it takes the network to deliver the packet once it is sent until it is received by the receiving side. Network delay is usually determined by the transmission propagation delay based on the distance between communicating hosts and the processing delay of intermediate repeaters and routers, including the time the packet spends waiting in processing queues at routers.

Packet Delay Variation (or jitter) is the random or deterministic variation in network delay. Random variation can be cause by thermal noise or other interference blocking transmission, causing network delay. Deterministic jitter can be caused by protocol design specifications: for example, in Wi-Fi's Collision Avoidance mechanism, the random back-off time can cause a packet to wait longer than expected, thus introducing jitter.

In a paper by Johnson, Balakrishnan and Tang [76], the latency created by the 802.11 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) Medium Access Control (MAC) scheme is analysed. In this scheme a node with data to send needs to listen for carriers on the channel that it would like to transmit on for a period called the Distributed Coordination Function (DCF) Inter-Frame Space (DIFS). If the channel is free for that period of time, it sends a Request-To-Send (RTS) on the channel, specifying how much data it wants to send. The Access Point then replies with a Clear-To-Send (CTS) to allocate the channel to the requester for the specified time required to transmit. This process takes time to complete and therefore adds network delay to the transmission process.

The Collision Avoidance mechanism comes into play when two hosts transmit an RTS at the same time. Each host adds a short random waiting interval after a collision is detected. As the colliding hosts will probabilistically not have the same random interval, one of the colliding hosts will start the RTS / CTS process before the other,

thus resolving the collision. According to Khan et al. [79] these random waiting intervals can cause jitter, adding to the already existing network delay.

3.1.3 Bandwidth Constraints

In some physical network carriers, bandwidth may be constrained due to hardware limitations, energy considerations, spectral availability or interference levels. Atzori, Iera and Morabito [32] state that in Internet-of-Things (IoT) setups, *Things* will typically be restricted in computational and energy capacity. On the other hand, even high throughput protocols such as Wi-Fi 802.11ac will be limited in terms of available bandwidth in the ISM band, as well as by interference, in the extreme dictated by the Shannon-Hartley theorem for noisy channels [119].

In a paper by Chen and Kunz [47], they investigate the effect of network delay, packet loss and bandwidth constraints on the experienced latency between a low-power health monitoring system and a central server. They compare four IoT protocols (MQTT, DDS, CoAP and publish/subscribe UDP). They found that in all cases, the experience latency increases as network bandwidth decreases, with the MQTT protocols experiencing a rapid increase in latency. The reason for this increase in experience latency can be explained as follows: if the network bandwidth is less than the required upload bandwidth, the message can either be discarded or sent in the next time step. If the message is not discarded and the next time step also does not have enough bandwidth available, the message would need to be sent over a number of time steps, causing a large experienced network delay.

3.2 Effect of Adverse Network Conditions on State Consistency

In this section we will discuss the possible results of the following adverse network effects on the state consistency. This section can be seen as the hypothesis of our tests in chapter 4.

3.2.1 Packet Loss

Packet loss will result in not all of the state updates being transferred to all of the clients in the VE. If only one packet is lost, the client that would have received a neighbour update now only has the state from the previous update. As no new information is available,

it is assumed that the state does not change. If the neighbour does not move, there is no inconsistency. However, if the neighbour does move, the state between the client and neighbour is now inconsistent. In this way the receiving client still believes that the neighbour is one step behind. Once the next update arrives, the state inconsistency can be resolved if an absolute update is received successfully, bringing the states back into agreement.

The user will experience this as temporary latency and then a jarring "too fast" update as the states are brought back into alignment. In the worst case scenario, typically in peer-to-peer architectures, the lost packet could also cause neighbours to be unaware of each other and therefore not exchange further updates.

3.2.2 Network Delay and Network Delay Variation

Network delay will cause the update packet to arrive late at the receiving clients. This will also be experienced as latency by the user, but as long as the network delay is constant, the packets will still arrive in order and therefore still be applied in the correct sequence.

If there is delay variation as well as network delay, greater problems could arise: each packet could be delayed a different amount. One possible outcome is packet reordering: if the higher-latency older packets are overtaken by the lower-latency newer packets, the received order will be incorrect. Newer state updates will be applied first, and as old state update arrive, they would then be applied, leading to confusion and an incorrect final state. This problem also presents itself in peer-to-peer architectures if there is a great disparity between the network delay between hosts, leading again to confusion as to the newest/correct state.

3.2.3 Bandwidth Constraints

As we saw in the paper by Chen and Kunz [47], bandwidth constraints can cause the receiving application to experience latency. This is because there is not enough bandwidth available to transfer the needed packets in the given time and therefore the transmission is postponed to the next available opportunity.

In terms of the effect on state consistency, similar to network delay, bandwidth restrictions can be experienced as latency. In virtual environment applications, the bandwidth requirements are usually related to the number of users of the VE. Therefore the latency experience would increase if more player join the VE than can be handled by the available bandwidth.

In the next section we will discuss two transport protocols, TCP and UDP, which are typically used to deliver state updates between neighbours.

3.3 TCP and UDP

We have selected TCP and UDP to carry the state updates in our network for the following reasons:

- TCP and UDP carry most of the Internet traffic [125] and is therefore the industry standard and well developed and implemented.
- TCP and UDP represent fundamentally different transmission paradigms: TCP is a reliable stream protocol designed to compensate for network conditions. In contrast, UDP is a simple datagram protocol, exposing the application to network conditions.
- TCP communication was already implemented in VAST. However, according to [121] most MMOGs use UDP, and therefore we decided to evaluate the effect of adverse network conditions on UDP as well.

TCP [41] is a connection-orientated and reliable protocol, providing ordered stream communication. TCP is used when high reliability is required, for example in file transfers or web applications. TCP sets up a connection with a three-way handshake, a three packet sequence number synchronisation interaction. Once completed, data transfers can start.

TCP transmits information as a stream of bytes, with variable packet lengths depending on the flow control algorithm. In order to provide reliability, the TCP stack needs to keep track of bytes sent successfully, with the endpoint using acknowledgments with sequence numbers to indicate successful transmission. If no acknowledgment arrives or some bytes are lost in transmission the sender will retransmit the needed bytes. Due to the larger TCP header, acknowledgments and retransmissions, TCP transfers will typically use more network bandwidth compared to UDP.

TCP flow control attempts to send bytes at an ideal rate such that the network can accept the load. If a packet is lost, the flow control algorithm interprets this as a symptom of too high transfer rate and reduces the transfer window size. If no packet loss occurs, the flow control algorithm will attempt to increase the window size

to an optimum. As the tests are conducted on a Linux system, the TCP congestion control algorithm is CUBIC [3].

UDP [101] is connectionless, message orientated without reliability or ordering guarantees. UDP is a simpler protocol than TCP, containing only source and destination ports, packet length and a checksum. The checksum is optional in IPv4. UDP does not establish a connection (i.e. no handshaking) and does not provide re-transmissions. The small header size, lack of handshaking and no retransmissions allow UDP to be set-up more easily and transmit data as soon as possible. UDP is typically used where latency is most important and some packet loss is acceptable.

3.4 Summary

In this chapter we discussed the possible adverse network conditions, reasons for why adverse network conditions appear and what effect it could have on the state consistency. Finally, we discussed why we will be using UDP and TCP as the network implementations for our tests on adverse network connections.

In the next chapter we will show the results of packet loss and network delay tests.

Chapter 4

Results of VAST MMVE Under Adverse Network Conditions

This chapter shows the performance of VAST under various network conditions.

First we will explain the Mininet network emulator and the network setup used for evaluating VAST. Then we will explain the metrics used to compare different network implementations. We will explain how tests were run on our testbed and how simulation parameters were chosen.

We will show the performance of VAST for varying packet loss percentages and verify the packet loss results using a real network and physical hosts. Then we will measure VAST performance under varying network delay and bandwidth constraints conditions. Finally we will test scalability at different node counts.

4.1 Mininet Emulator and POX Controller

In order to evaluate the performance of VAST under various network conditions, we need a network environment for which we can control conditions such as packet loss, network delay and bandwidth limitations. We also want a network environment which emulates a physical network as closely as possible in order to evaluate VAST under conditions that are close to realistic conditions.

Mininet [9] is a network emulator using the Linux kernel and light-weight virtualisation. Mininet uses network namespaces to separate processes on the same host, emulating the separation of hosts without the need for multiple physical hosts. Communication between network namespaces is achieved by virtual Ethernet interfaces. These interfaces can be controlled by the Linux Traffic Control

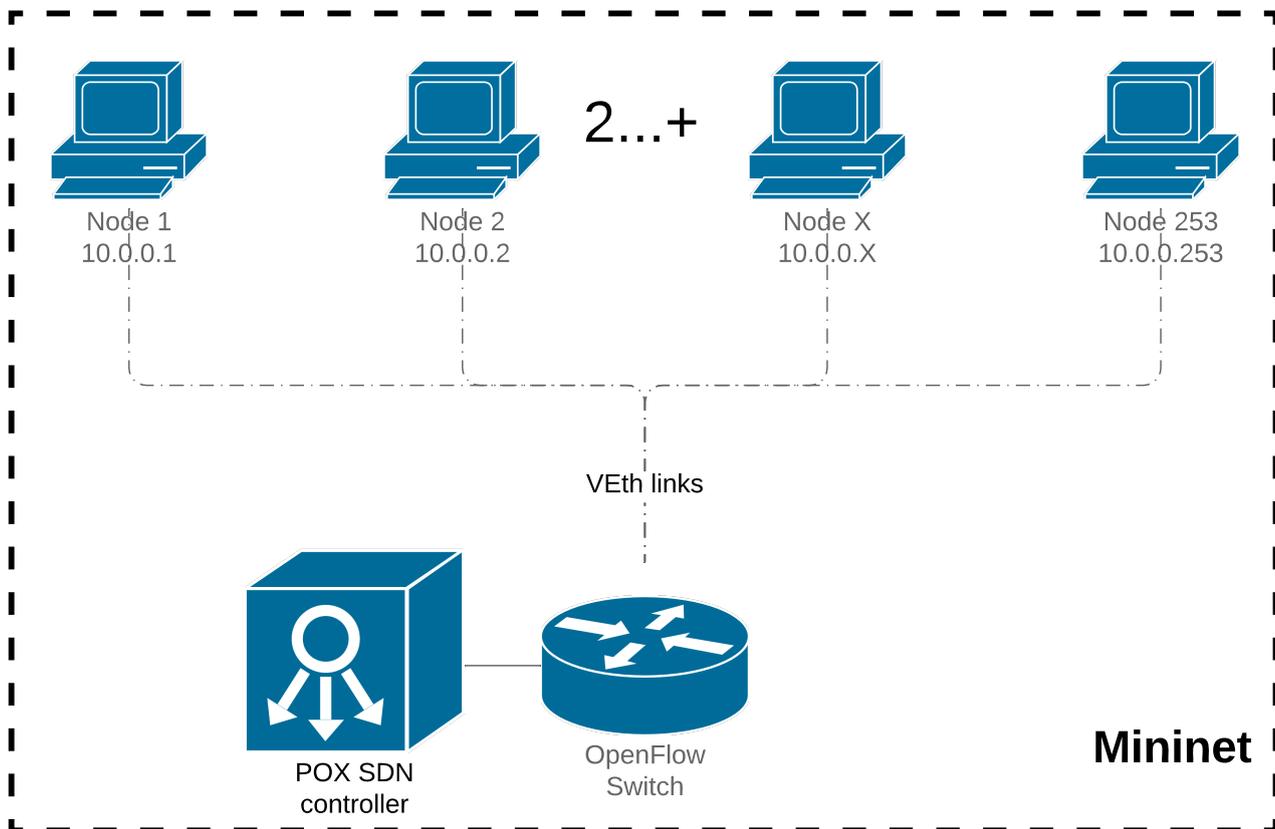


Figure 4.1: Mininet network setup using a POX controller.

utility `tc`. For more details please see their Introduction to Mininet section [5].

Mininet provides all of the tools needed to emulate real Ethernet networks using lightweight virtualisation. Programs run on a Mininet emulation can be used directly on a real network without changes to the implementation.

Mininet implements the Software Defined Networking (SDN) paradigm by using OpenFlow [13] enabled network switches. Switching rules/tables can be customised using the OpenFlow protocol, allowing Mininet to be used in a wide variety of applications and scenarios.

The POX controller [21] is a Python program which controls the switching tables on the OpenFlow switch. In our experiments we use a simple layer 2 learning controller. This means that packets will be switched based on their Ethernet MAC address. Hosts have both an unique Ethernet address and IP address.

During the operation of the network, the POX controller will learn

how the output ports are connected to the destination hosts, mapping the output ports to their Ethernet addresses.

Initially, when no address-to-port mapping is known, the packets are flooded to all connected hosts. As hosts send packets on the network, Ethernet source addresses are used to create a address-to-port mapping. Future packets can then be switched to the correct output port based on the Ethernet destination, instead of flooding the network. Layer 2 Learning is the default operation mode of OpenFlow switches.

In order to translate the IP addresses to Ethernet addresses, individual hosts keep a table of known IP-to-Ethernet address mappings. If a packet needs to be sent to a unknown IP address, the Address Resolution Protocol (ARP) [17] is used: an ARP packet is broadcast to all hosts, asking which host corresponds to that IP address or which Ethernet address is the gateway for that IP. The host with the requested IP replies with its Ethernet address. The packet can then be dispatched to the switch with the correct destination Ethernet address. The switch is responsible for forwarding it to the Ethernet address specified.

The protocols and network setup are all standard practice for a local area network implemented using the SDN paradigm.

4.2 Network Setup

In Figure 4.1 we can see the network setup used in our evaluation. Each of the nodes are separated by Mininet in a separate network namespace as was described in the previous section. Nodes are addressed by a static IP. All of the nodes in the VE are connected to the SDN switch. Each node runs the VAST executable.

Note that only 253 IP addresses can theoretically be used in this subnetwork as we use a 255.255.255.0 netmask and we reserve one IP for the coding host. The coding host is a special host which forms part of our network coding packet loss mitigation strategy.

However, we do not use more than 100 nodes in any of our simulations and therefore we will have enough IP addresses available.

Mininet allows the use of Python scripting to create custom topologies (such as is shown in Figure 4.1) and set link properties such as packet loss rates, network delay and bandwidth limitations.

4.3 Additions to VAST

As far as possible, the VAST library was kept unaltered in order to allow comparisons between this work and previous work by Hu et al. ([69], [71], [70]). The core interest matching and load-balancing functionality of VAST was used as is. The existing VAST implementation used the ACE library [1] and TCP as a transport protocol.

However, in order to evaluate the performance of VAST on the Mininet emulator, we made the following additions to VAST:

- The original TCP implementation assumed that all operations would occur on the localhost interface. We expanded the implementation to run VAST on a network with different IPs, allowing an evaluation on the Mininet emulator and physical networks.
- As UDP is typically used for small game state updates [121], we also implemented a UDP network component for VAST using Boost [2] libraries in order to evaluate VAST performance using UDP as a transport protocol.
- Finally, as the executables run on separate Mininet hosts instead on a single localhost process, the time series state of each node needed to be recorded to a log file. The Boost serialization library was used to store state in log files. After the experiment was completed, the log files were deserialised and merged to form the global state. Consistency calculations, as was implemented by the original VAST implementation, were performed.

Appendix A explains the details of how state and network variables were recorded to a log file and how the deserialisation, merging and consistency calculations are completed. However, a detail understanding of the log files are not needed to understand the consistency metrics used in this chapter.

The full source code is available at:

https://github.com/dschoonwinkel/VAST_NC.

4.4 Real versus Simulated Network Stack

As was mentioned in the previous section, the original VAST implementation did contain a TCP implementation. However, most of the larger simulations in Hu's previous work were run using a simulated network stack: instead of packets on a NIC interface, packets moved between memory locations to simulate transmission. All nodes also run in a single process such that all nodes shared a memory space

and all computation was performed in the same thread. This drastically reduces processing load and avoids the stochastic nature of multi-threaded systems.

In contrast, all of our nodes run in independent Linux processes and use full network stacks. Because of the independent processes, the update steps are not synchronised.

In practical VE scenarios, synchronisation is difficult as the nodes are distributed. Furthermore, as users are separated and use different hosts, the VE scenario will inevitably have separate processes and use full network stacks to communicate over the Internet. Therefore our Mininet simulations are more comparable to a practical VE than the original VAST implementation with simulated network stacks.

Using a real network stack will also yield a better understanding of the bandwidth requirements of such a VE application, including the bandwidth added by the protocol overheads. In our simulation, VAST only communicates the movement updates. Therefore the bandwidth requirements are very low. As a order of magnitude test, we calculate the expected bandwidth of only the movement updates. Each movement update packet is between 60 - 100 bytes (as was observed during operation) and node communicate one update per 100 ms timestep. That results in an expected bandwidth between 600 - 1000 Bps per node from movement updates alone. The average sending bandwidth in the network will be higher, because these movement updates will need to be sent by the matcher to all the interested clients. However, this gives us an order of magnitude estimation of the bandwidth requirements.

It is clear that this application has low bandwidth requirements and is not comparable to streamed VR or cloud-hosted game streaming. However, in this project we assume that only the state of the VE is sent and a local machine or VR device renders the world. From the rough bandwidth estimation above we can see that our intended application (VR, AR or MMVE over a local connection) would suit the bandwidth requirements.

In literature the bandwidth requirement of a client-server based game is given as between 250 Bps [104] and 7.5 kBps [97].

4.5 Consistency Metrics

Hu et al [69] defines two consistency metrics: topology consistency percentage and drift distance. Topology consistency measures the ability of nodes to interact with each other. Drift distance measures the accuracy of each node's local state compared to the global state.

These two consistency metrics are calculated in VAST simulations, and also in our experiments. Using these two metrics, we can compare the state consistency of user positions at each step in the simulation. We define a step as one tick of the simulated time clock or one event / update opportunity. A step can also be compared to a frame of video game.

In order to allow repeatability, users are moved in simulated paths.

4.5.1 Topology Consistency

Hu et al [69] describes topology consistency as the percentage of known neighbours compared to the actual neighbours, calculated at each step. Topology consistency is calculated as follows:

$$TC\% = \frac{\sum_{k=1}^K N_k}{\sum_{k=1}^K M_k} \times 100 \quad (4.5.1)$$

where K is the total number of nodes in the VE, N_k is the known AOI neighbours and M_k is the actual neighbours at node k that falls within its AOI.

4.5.2 Normalised Drift Distance

Drift distance is defined as the difference from the perceived position of a neighbouring node and the true position of the node. The perceived position of a neighbour could be in error due to packet loss or latency. We normalise the drift distance by the number of nodes experiencing drift in order to make drift distance at different simulation steps comparable. Normalised drift distance is calculated as follows:

$$NDD = \frac{\sum_{k=1}^K \sum_{n=1}^N (\text{dist}(PP_{n,k}, AP_{n,k}))}{\sum_{k:DD_k \neq 0}^K 1} \quad (4.5.2)$$

where N is the known AOI neighbours, dist is the Euclidean distance function in a 2D plane rounded to the nearest integer, $PP_{n,k}$ the perceived position of the neighbour n at node k , $AP_{n,k}$ the actual position of the neighbour and K is all the nodes in the network.

$\sum_{k:DD_k \neq 0}^K 1$ counts the number of nodes experiencing drift at the current step. The unit for this measurement is in-game world distance units.

Topology consistency and drift distance are correlated - a lower topology consistency will typically coincide with a higher drift distance. However, drift distance is more sensitive to adverse network conditions than topology consistency. For example a single packet loss will cause an immediate drift distance, whereas some packets can be lost before the topology will become inconsistent.

Note that the above metrics only describe the consistency of position updates. In a practical VE, state would include many other variables concerning the state of players, entities and the environment itself. The position updates are frequent and therefore state inconsistency due to adverse network effects can be resolved quickly (as soon as the next update arrives). In contrast, with infrequent environments updates such as the state of a door opened by a player, adverse conditions could cause an inconsistency for significant time. This type of state consistency should also be considered in practical VEs, but is considered beyond the scope of this dissertation, but should be included in future work.

4.5.3 Average VAST Latency

Latency is defined to be the elapsed time between the occurrence of an event and the response to that event. In networks, latency is typically defined as the time interval it takes for a packet to reach its destination or to receive a reply response. In this dissertation we will measure latency of the *MOVE* VAST messages. *MOVE* message indicate that a client has changed its position and AOI. The *MOVE* message is sent from the client to the Matcher and is then forwarded to the interested subscribers (i.e. neighbours). On receiving the *MOVE* message, the client records the latency.

In our results we only use *MOVE* message latency, as this will influence the VE experience the most.

The average latency is calculated as follows:

$$L = \frac{\sum_{i=1}^R (t_r - t_s)}{R} \quad (4.5.3)$$

Where L is the latency, t_s the clock time at message creation and t_r the clock time when receiver processes the message. R is the number of latency records.

All hosts in the Mininet emulation share the same clock as they use the same Linux kernel instance. When using physical hosts, clocks can be synchronised by using the Network Time Protocol [15].

4.6 Network Performance Metrics

Consistency metrics are needed to evaluate the state consistency. In addition, we also want to evaluate how effectively the network resources are used. In this section we will explain the measured network bandwidths and how they can be used to indicate network load.

4.6.1 Normalised VAST Send / Receive Bandwidth

The first network metric was built into the VAST library. The number of bytes sent and received by the VAST network implementation is recorded for each step. The VAST byte counts represents the payload bandwidth, i.e. the bandwidth without the transport and network layer headers.

By multiplying the update frequency by the VAST send / receive bandwidth, we can get to a value of kilobytes per second. We then normalise this bandwidth with respect to the number of nodes in the network to get to a value in kilobytes per second per node.

The normalised VAST send / receive bandwidth is calculated as follows:

$$\text{NVBW} = \frac{\sum_{i=1}^K \text{VBW}_i}{K} f_{\text{step}} \quad (4.6.1)$$

where VBW_i is the VAST send or receive bandwidth of node i at the current step, K the number of nodes and f_{step} the update frequency.

Unlike the latency measurements above, this metric considers all VAST messages.

4.6.2 Normalised NIC Send / Receive Bandwidth

The send and receive bandwidths moving through the Network Interface Card (NIC) is also measured to determine the actual needed bandwidth for communication. This bandwidth includes the overhead for the headers of the network protocols (either TCP or UDP). The NIC bandwidth is measured externally from the VAST executable using the `tshark` commandline utility. Similarly to the VAST bandwidth, this bandwidth is normalised by the number of nodes in the network.

The normalised NIC send / receive bandwidth is calculated as follows:

$$\text{NNBW} = \frac{\sum_{i=1}^K \text{NBW}_i}{K} f_{\text{step}} \quad (4.6.2)$$

where NBW_i is the NIC send or receive bandwidth of node i at the current step, K the number of nodes and f_{step} the update frequency.

4.6.3 Preference to NIC Send and NIC Receive

As we are interested in investigating the network implementations, we are more interested in the actual NIC bandwidth than the VAST bandwidth.

In Figure 4.2 we show the VAST send and receive bandwidth as a reference to show how information is generated for the network. We also show the NIC send and receive bandwidth to indicate how the network experiences the load. It can be clearly seen that VAST bandwidth is strongly related to NIC bandwidth. In essence the only difference is the extra bytes added by network encapsulation included in NIC bandwidth but not taken into account with VAST bandwidth. We therefore propose that a thorough understanding of the network conditions can be achieved with only NIC bandwidth and the VAST bandwidth will not add significant insights.

Thus even though VAST bandwidth is recorded, it will not be included in results.

4.6.4 Send and Receive Bandwidth Equality

From Figure 4.2 it can be seen that receive bandwidth is equal to send bandwidth. That is, the send bytes will be recorded by the sending side and the receive bytes will be recorded by the receiving side, and the normalised bandwidth should agree unless there is a failure in the network.

In Figure 4.3 we can see what happens if there is a network failure such as packet loss. The VAST bandwidth remains the same, but the NIC receive bandwidth is 6.24 kBps per node, whereas the NIC send bandwidth is 6.93 kBps per node. Packet loss causes roughly 10% of the packets to be dropped, which is the packet loss value which was used in the Mininet emulator for this run.

4.7 VAST Components

In order to ease discussion of the simulation parameters and results, we remind the reader of two VAST roles as was discussed in section 2.8.

The *Matcher* is a super-peer responsible for managing the spatial subscriptions and forwarding spatial publications to the correct subscribers.

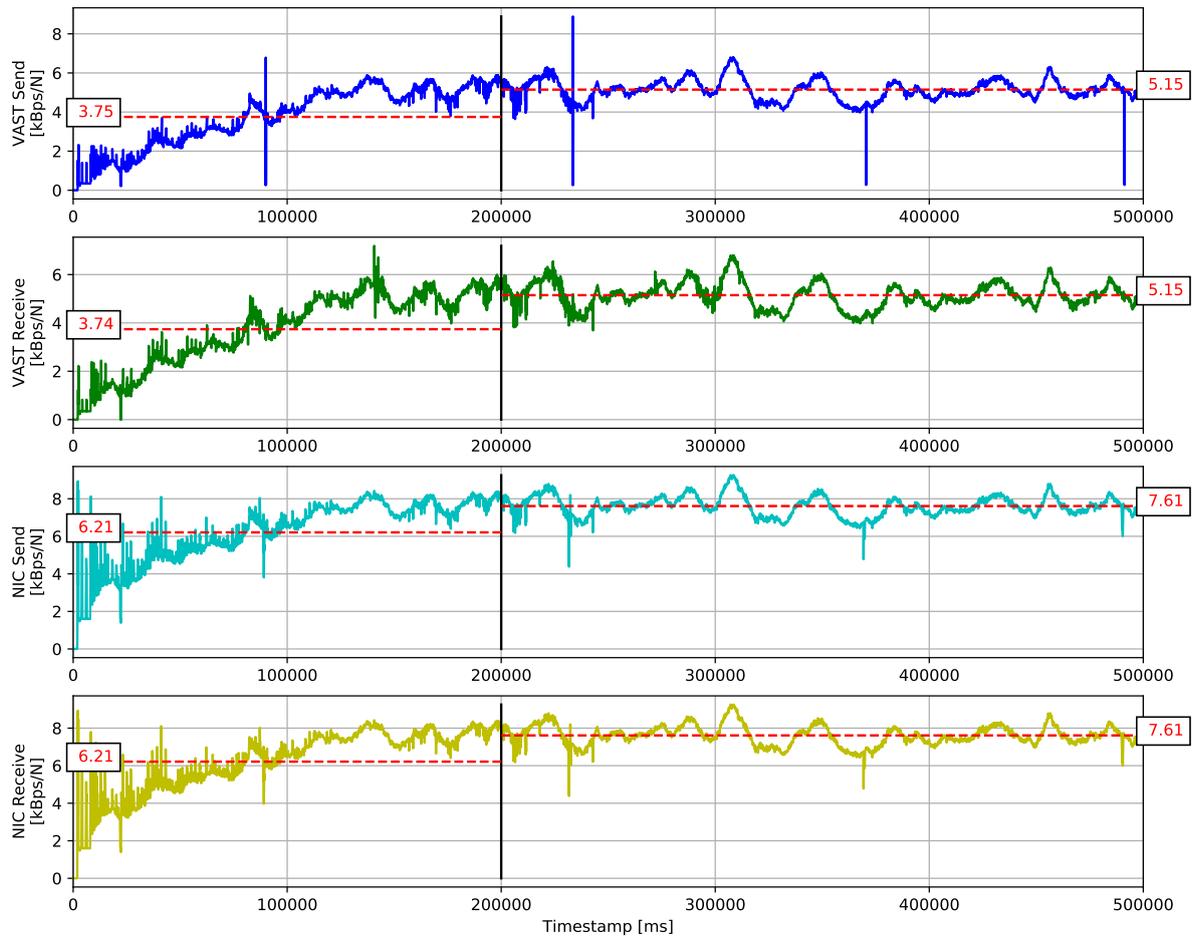


Figure 4.2: VAST and NIC bandwidth comparison with no packet loss. The black line indicates the end of the setup phase, meaning that the network has reached steady state. From this point onwards, all the nodes in the VE are in the *JOINED* state. As there is no packet loss in this run, no change in network conditions are applied. Averages of the setup phase are given on the left, and averages after the setup phase are given on the right.

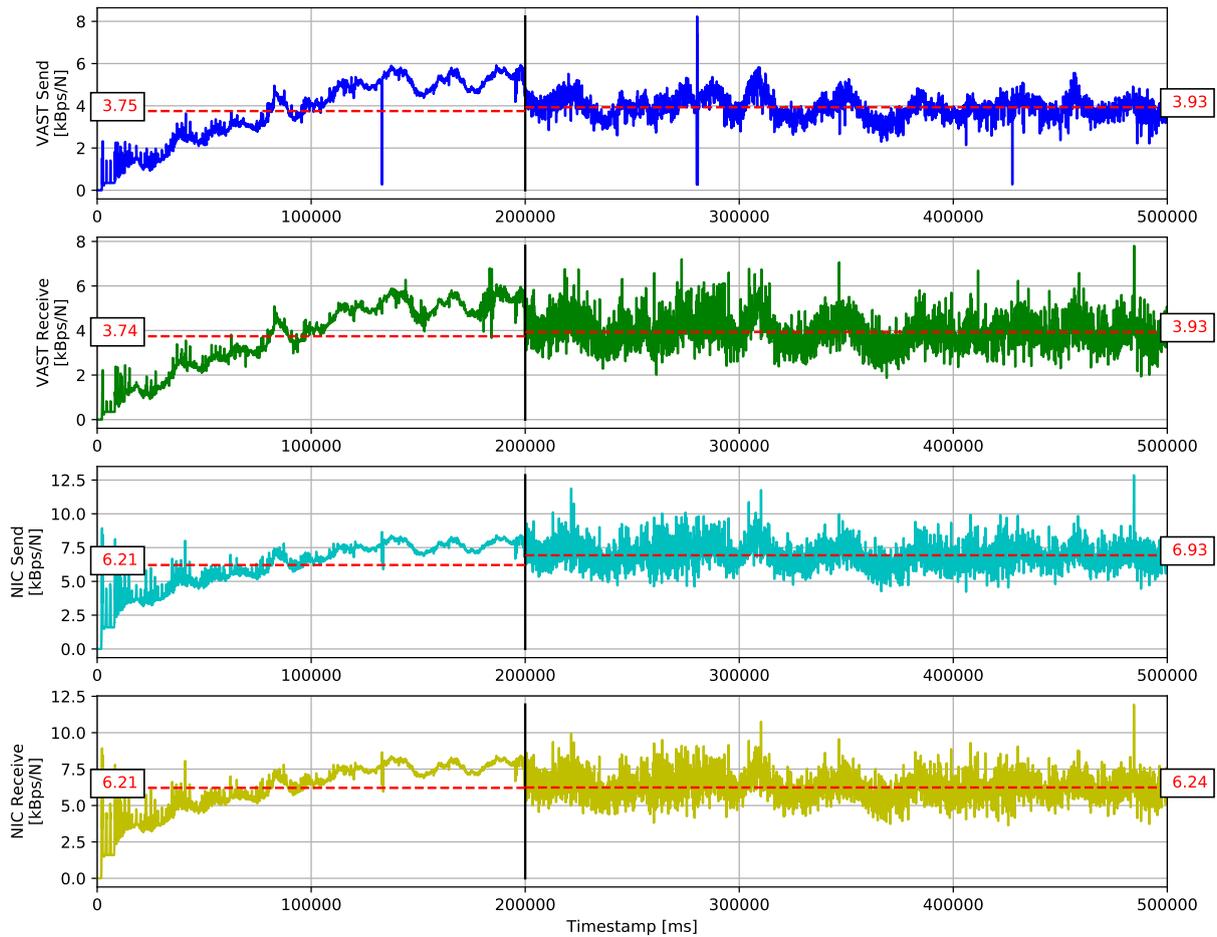


Figure 4.3: VAST and NIC bandwidth comparison with 10% packet loss. The black line indicates the end of the setup phase, meaning that the network has reached steady state. From this point onwards, all the nodes in the VE are in the *JOINED* state. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. Averages before packet loss are given on the left and averages after packet loss is applied are given on the right.

Table 4.1: General simulation parameters

Parameters	
World dimension (units)	768 x 768
AOI radius	195
Step duration	100 ms
Simulation setup steps	$10 * K + 1000$
Total Simulation steps	5000
Join rate	1/s or 1/10 steps
Speed (units / step)	3
Movement model	Uniformly random
Number of matchers	1
Iterations	20

The *Client* is a normal peer in the VE. Clients (representing users) move around in the VE and can subscribe to areas they are interested in. Note that a node can host both a matcher and a client.

4.8 Simulation Parameters

Similarly to Hu et al. in [70], we use a world size of 768 x 768. These dimensions come from the Second Life representation of an area of size 256 x 256, which has been enlarged with a factor three for better visualisation. Similarly, the AOI is 64 (from Second Life) times three, but increased slightly to ensure that AOIs overlap even in edge cases.

For most of our tests, we chose a node count K of 50, similar to the lower value of Hu et al's paper. It is important to note that our simulation uses Mininet nodes and networking which better represents real networking. Hu used emulated networks in their simulations which use significantly less CPU resources. We therefore decided to use the lower bound so as not to influence our simulations with overloaded CPU effects.

In order to allow all of the nodes to connect to the VAST overlay, we allow 10 steps per joining node plus 1000 steps as our simulation setup duration. No adverse network conditions are applied during the setup phase. During the setup phase, overlay join messages are sent and neighbour discovery happens. After the setup phase, all of the nodes should be in the *JOINED* state and only be communicating their move events. The setup duration can also be described as the settling time and the rest of the simulation as the steady state.

The total simulation length is set at 5000 steps. If we have 50 nodes joining the VE, the setup phase is $50 \times 10 + 1000 = 1500$ steps.

The adverse network conditions are applied after the setup steps, so that we have 3500 steps of operation in such conditions, comparable to Hu's 3000 steps.

The step duration was set at 100 ms or 10 Hz update frequency. This ensures that state update calculations can be completed within the update step duration.

In a highly interactive application, such as a first-person shooter game, the slow update speeds could affect user experience [34]. In this case, faster update speeds should be used in a real-world test, and more processing power would be required so as not to limit interactivity. In our tests we used 100 ms step duration, with faster updates proposed for future work.

Movement speed was set to 3 units per step, similar to Hu's study. Although literature suggests that real movement patterns significantly differ from random walk movement (eg. [83], [40]), we use random walk movement, as was originally implemented in VAST. Using a movement model already included in VAST allows the comparison to other papers also using VAST. However, performing tests using traces from, for example, recorded Second Life movement patterns would increase the accuracy of the results and is considered future work.

The join rate is 10 times slower than Hu's study, but a slower join rate is needed as we our project includes a full network stack which can influence the success of joining.

The simulation was limited to one matcher. This eliminates the extra complexity of multiple Matchers communicating to each other. It was also observed that inter-matcher communication is very sensitive to packet loss and therefore more matchers greatly reduce the consistency. Future work should include more reliable inter-matcher links, possibly with strongly increased redundancy for such communications.

Each configuration was run 20 times, allowing a reasonable analysis with the median and 90% confidence intervals. Having data from 20 runs also allows us to identify outliers.

4.9 Nominal Network Conditions

Firstly, we compare the network implementations in nominal conditions (no packet loss, no delay, no bandwidth limitations) to determine the baseline performance.

4.9.1 Experiment Setup

Each network implementation was used to run a simulation of 5000 steps with 50 nodes in the network. Without adverse network conditions, this setup is the ideal case for any network application. The nodes are started by the Mininet topology script as specified by the join rate. The first node will be promoted to the matcher role, therefore having both the matcher and a client. All other nodes will only have a client that connects to the first node's matcher. When the simulation is finished (5000 steps completed), the Mininet topology script will stop all the VAST processes and hosts. The consistency logs will be analysed and results written to file. The mean of each run will be computed from these result files and plotted using boxplots.

4.9.1.1 Boxplots

For clarity, the boxplots are constructed as follows: The box ends represent the first (Q1) and third (Q3) quartile points. The whiskers are drawn based on the Inter-Quartile Range (IQR): the top whisker is the largest value that still falls within $Q3 + IQR \times 1.5$. The bottom whisker is the lowest value that still falls within $Q1 - IQR \times 1.5$. Values outside of the whiskers are considered outliers and are indicated by black circles.

4.9.2 Results

Figure 4.4 shows that both network implementations achieve near perfect topology consistency. This is expected as there is no packet loss and all nodes are aware of their neighbours. As the update steps of nodes are not perfectly synchronised, small inconsistencies can occur while a node waits for neighbour update steps and transmissions. These small inconsistencies are resolved within one update step, but in some case prevent all nodes to be aware of their neighbours at exactly the moment two AOIs overlap.

It can be observed that the drift distance and latency of UDP is slightly higher than TCP. In contrast, the TCP send and receive bandwidths are higher than UDP.

4.9.3 Discussion

Due to the larger header size and acknowledgments used in TCP, the bandwidth use will be larger compared UDP. As there is no adverse network effects in this experiment, TCP has slightly higher topology consistency than UDP. The lower drift distance and latency

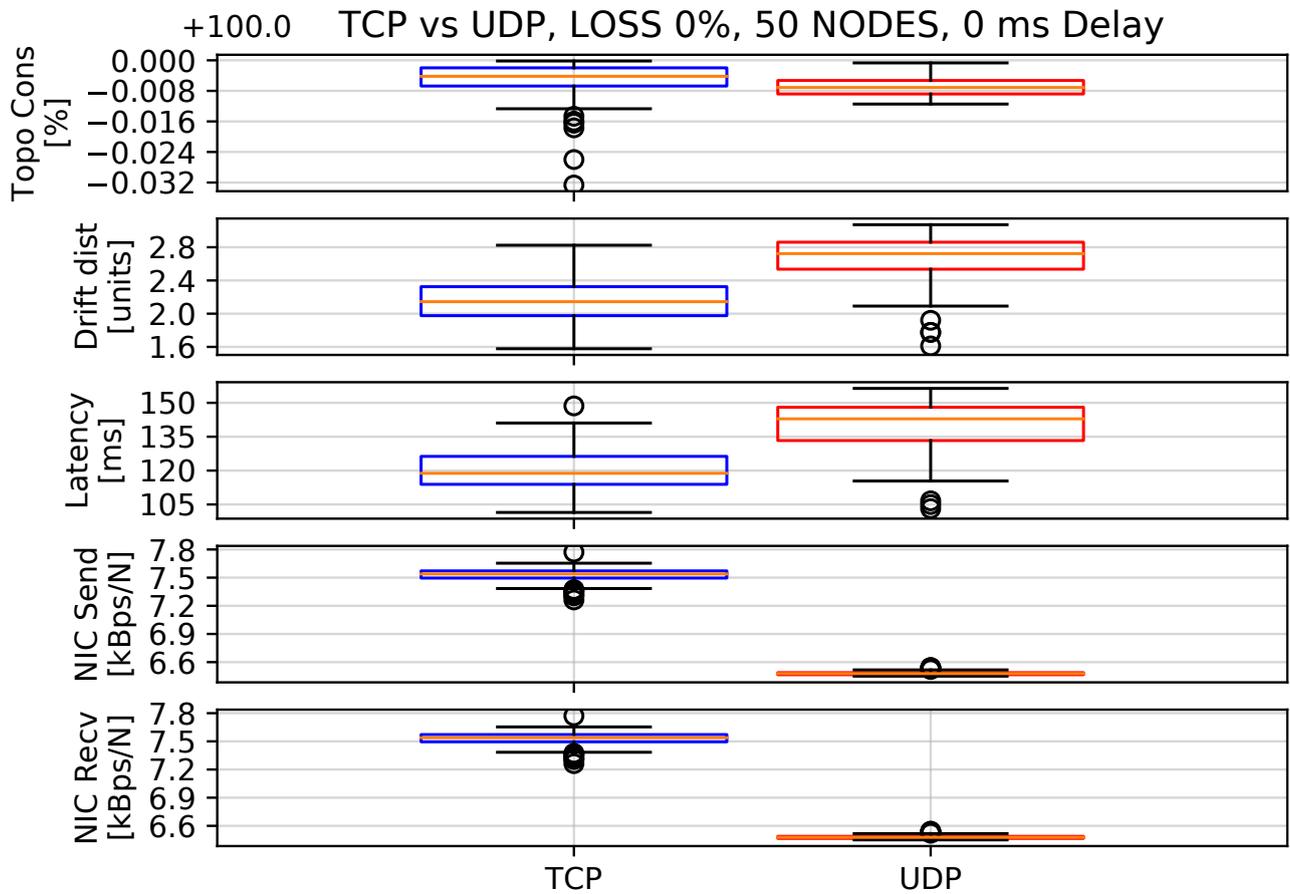


Figure 4.4: Nominal network conditions. Aggregate topology consistency, normalised drift distance, average latency, and normalised NIC send and receive bandwidths are shown for TCP (blue) and UDP (red) tests. Black circles indicate outliers.

in TCP can be explained by the implementation differences: TCP uses a process thread for each connection to another node. In contrast, UDP packets are received on the same interface and is therefore processed by the same thread. This single-threaded nature of UDP causes a slight delay in processing UDP packets, resulting in a higher latency and therefore also a higher drift distance.

4.10 Adverse Network Conditions Testing

In order to understand the effect of adverse network conditions, we will investigate the effect of packet loss, network delay and bandwidth limitations on the state consistency of the VE *separately*. Note, however, that these effects are typically related in practical networks.

In wired networks, where physical layer loss is typically low, network delay and packet loss is the end result of buffer overflows (which can be seen as a maximum bandwidth constraint): high or bursty traffic cause buffers to fill up at routers, resulting in increased network delay for the packets in the buffer. In the extreme case, the buffers can completely fill up and cause packet loss due to buffer overflows [57].

In contrast, in wireless networks packet loss is primarily due to interference on the hardware layer. In networks where hardware layer retransmissions are enabled (such as Wi-Fi 802.11), packet loss results in retransmissions, in turn leading to longer delays for successful transmission. Retransmissions also limit the number of new packets that can be sent by a router, thereby placing a bandwidth limitation on the link as well [29].

Firstly, we will show the effect of packet loss on the VE consistency. We will also verify the packet loss test result obtained from Mininet simulations by evaluating VAST performance on a real network using physical hosts. According to Sheshadri and Koutsonikolas [110], using 802.11n Wi-Fi without MAC layer retransmissions can result in packet loss rates varying between 10% - 80% depending on the Modulation and Coding Scheme (MCS) used. It is not always possible to disable MAC retransmissions as the MAC layer could be implemented in hardware and therefore in practice MAC retransmissions would be enabled by default. However, in Mininet we are not able to emulate the mechanics of the MAC retransmissions and therefore we perform simulations as if they are disabled. Our simulation therefore probably assumes too high packet loss rates for wireless and do not account for retransmission delay. However, in

the delay tests, the effect of possible retransmission delay is evaluated.

In the subsequent delay and scaling tests we will observe the state consistency under 10% packet loss in order to represent the lossy conditions of a wireless network. Using 10% packet loss (the lower bound on packet loss) in the delay and scaling tests aligns with our purpose of investigating the effect of adverse network conditions on the VE state consistency, potentially using wireless, lossy communication.

In the bandwidth tests we will not use the 10% packet loss as the bandwidth limitations are already very restrictive and other adverse network conditions would prevent the VAST overlay to operate at all.

4.11 Packet Loss Tests

As was discussed in Section 3.1.1, packet loss is prevalent in wireless communication. In this section we will test the effect of various packet loss rates on the state consistency of a VAST environment.

4.11.1 Experiment Setup

In this experiment, a normal VAST simulation is run as was specified in section 4.9.1. A Mininet topology was used to specify the packet loss rate. Packet loss is only applied after all of the VAST clients have joined the VE and the VAST overlay has reached steady state.

Note that packet loss is applied on downstream links only. That is, we apply the packet loss on data moving from the switch to the nodes and keep the data flow originating from the nodes destined for the switch lossless. Applying packet loss on only one of the links is comparable to applying half the value of packet loss on both links. We therefore do not alter the simulation significantly by applying packet loss only to the downstream links. However, the proposed packet loss mitigation strategy (presented in Chapter 6) is only able to recover packets on the downstream links and therefore we run the current tests with downstream packet loss in order to make the later results comparable with this chapter's results.

Mininet applies packet loss using the command line utility `tc`. The documentation for `tc`, specifically its network conditions emulator can be found here [7]. In this network emulator, each processed packet is assigned a random probability to be lost in transmission, according to the packet loss percentage required.

In addition to the simulation parameters given in Table 4.1, this specific test uses the simulation parameters given in Table 4.2. We

Table 4.2: Packet loss test simulation parameters

Packet loss	0-70%
Number of nodes	50
Delay	0 ms

chose a maximum packet loss rate of 70%, similar to Gheorghiu et al's paper. This is a reasonable upper limit as transmission becomes very difficult for any protocol above this value.

4.11.2 1-2-5 Series as Preferred Numbers

The 1-2-5 series is a set of preferred numbers, based on the aggressively rounded E3 series. The E3 series spaces three points logarithmically in a decade. The E series of numbers was originally defined in [72]. In typical electrical components, the E3 series would represent the values 1, 2.2 and 4.7, typically with 20% tolerances allowed. When these numbers are rounded further they come out to 1, 2, 5, yielding 1, 2, 5, 10, 20, 50, etc as preferred numbers. Our choices of packet loss percentages and delay is based on these preferred numbers, with extra points in-between when deemed necessary.

For better understanding our data, we first look at a single run of each of the two implementations: TCP and UDP.

4.11.3 Single Run Analysis: TCP

Figure 4.5 shows the measured values during a TCP run at 10% packet loss.

The top most graph shows the increase in active clients in the VE. The clients connect sequentially. The join process is started according to the join rate, but it could take some time for the node to reach the *JOINED* state. This causes the irregular joining seen in the top most graph.

After all the clients have joined, there is a settling time before packet loss is applied. The black line indicates the end of the settling time.

Before packet loss is applied, topology consistency remains close to 100% with only small deviations. Such a high topology consistency means that most clients know their neighbours and only a few clients are unaware of some of their neighbours. Once packet loss is applied, a lower topology consistency of 99.26% can be observed, with more variance. This is a direct result of packet loss, causing some clients to be unaware of their neighbours. Note that TCP and

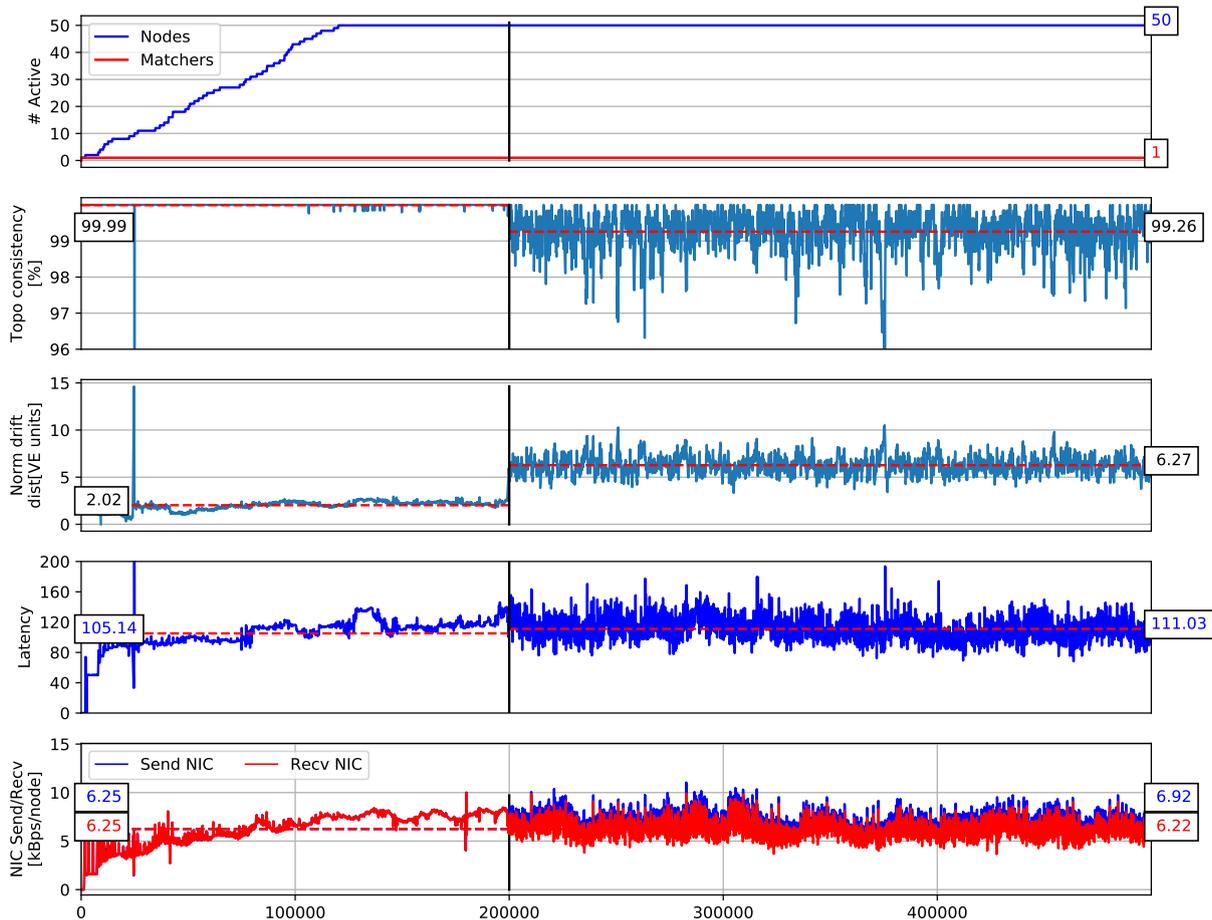


Figure 4.5: Results of a TCP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. In this figure the drift distance plot is scaled to the range of normalised drift distance for readability. See Figure 4.6 for full range of maximum drift distance. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right.

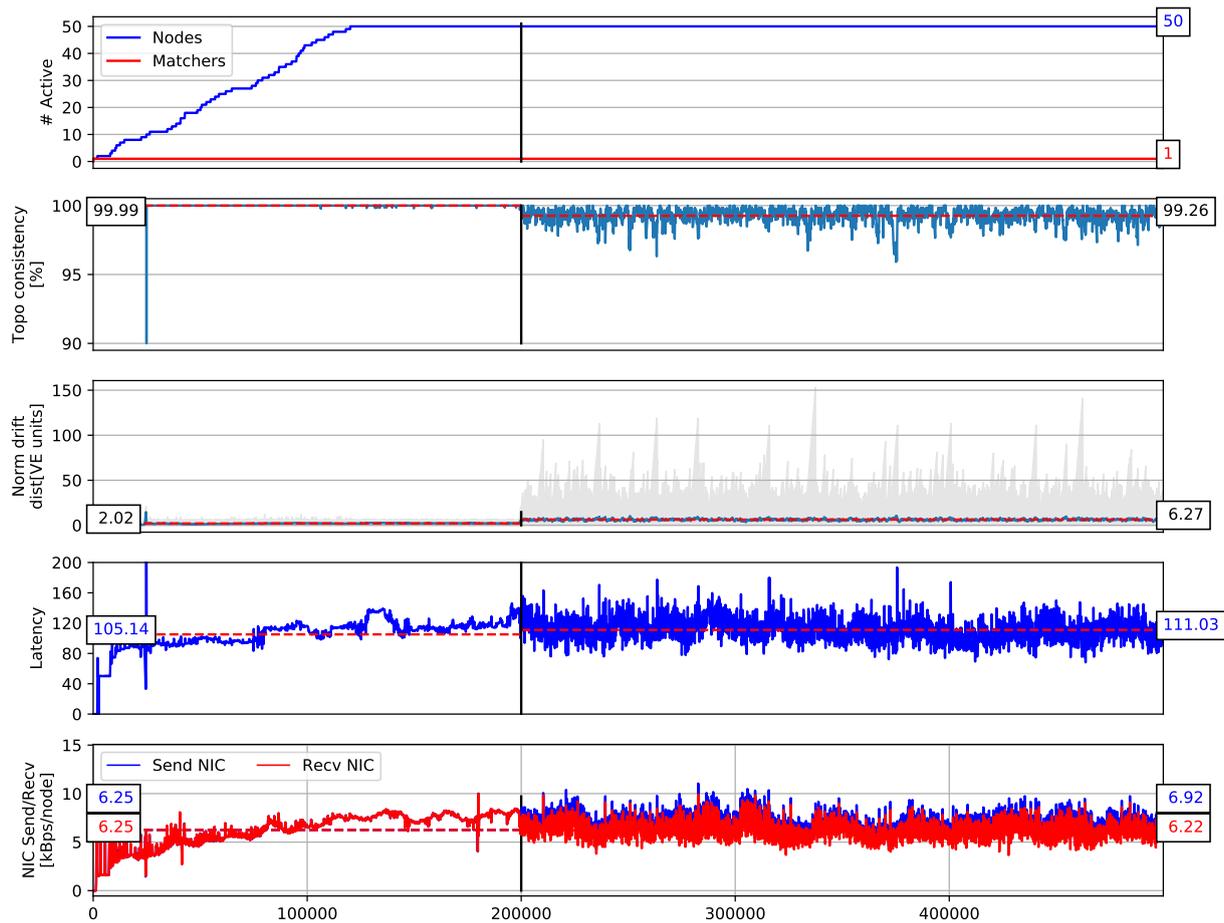


Figure 4.6: Results of a TCP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Topology consistency and drift distance plot shows the full range. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right.

UDP topology consistency graphs all use the same y-axis scale for comparison.

Drift distance remains fairly low at 2.02 VE units before packet loss. The moving speed of nodes are 3 VE units per second, therefore the clients are on average less than one step behind the actual position. However, this increases significantly, to 6.27 VE units, when 10% packet loss is applied. Such a drift distance can be described as effectively two steps behind schedule. The maximum drift distance observed on a node in the network is indicated by the light gray background.

Latency remains roughly the same, with a noticeable increase in variability, probably due to the retransmissions of TCP. Variations in latency correspond well to the variations in drift distance.

In the last graph, we show the normalised NIC sent bandwidth (blue) and NIC received bandwidth (red). Once packet loss is applied, NIC received bandwidth becomes more variant, due to lost packets. The bytes are still sent, but less bytes reach the other nodes' NICs. TCP will attempt to retransmit packet to compensate for packet loss, leading to a higher NIC send bandwidth (shown in blue).

4.11.4 Single Run Analysis: UDP

Figure 4.7 shows the measured values during a UDP run at 10% packet loss.

Firstly, it can be noted from the number active nodes graph that joining over UDP is more regular than over TCP. A possible explanation is that UDP is a simpler protocol than TCP and therefore the overhead (including retransmissions) is less than with TCP. Furthermore, the TCP implementation starts a processing thread for each TCP connection, which could take time to set-up.

Similar to TCP, UDP topology consistency and drift distance performs nominally with no packet loss applied. After packet loss is applied, the topology consistency decreases to 99.84%, which is less degradation compared to TCP's 99.26%. This can also be explained by the simplicity of the UDP protocol: TCP retransmissions cause extra delay in packet lossy conditions, while UDP always forwards the latest packet as soon as possible. If a packet is lost, the clients recover quicker, due to the latest state being available sooner. This argument also applies to the drift distance: the latest update removes the drift distance immediately.

As with TCP, the light gray background in the drift distance graph shows the maximum drift distance.

Latencies are given on the same y-axis scale as TCP for comparison. Note that latency is measured on the application level and

Table 4.3: Median of average topology consistencies with 90% confidence interval, for increasing packet loss test.

	TCP	UDP	± %
0.0 %	100.00 ± 0.00	99.99 ± 0.00	-0.00 %
1.0 %	99.98 ± 0.00	99.98 ± 0.00	0.01 %
2.0 %	99.95 ± 0.00	99.97 ± 0.00	0.02 %
5.0 %	99.83 ± 0.00	99.93 ± 0.00	0.09 %
10.0 %	99.27 ± 0.00	99.84 ± 0.00	0.57 %
15.0 %	97.30 ± 0.11	99.69 ± 0.00	2.45 %
20.0 %	82.43 ± 0.17	99.51 ± 0.00	20.73 %
30.0 %	36.48 ± 0.35	98.80 ± 0.00	170.83 %
40.0 %	45.34 ± 0.25	97.23 ± 0.00	114.46 %
50.0 %	59.58 ± 0.21	93.81 ± 0.00	57.45 %
60.0 %	69.26 ± 0.10	86.60 ± 0.01	25.04 %
70.0 %	74.26 ± 0.09	71.04 ± 0.05	-4.33 %

therefore includes on the processing delay of the network and application. Latency average is higher than TCP: this is due to the single process implementation of UDP causing a processing delay due to queuing. However, the variability of UDP latency is less than TCP, because UDP does not perform retransmissions, limiting the variation to processing delays. It is important to note that if a packet is not transmitted successfully, the latency is obviously not recorded. Thus the UDP average latency is only calculated on received packets and whereas TCP average latency is calculated on successfully received and received retransmitted packets.

A lower NIC receive bandwidth than send bandwidth can be observed (due to packet loss).

4.11.5 Aggregated Results Analysis

In order to understand the general trend, we aggregate the steady state averages (shown at the right side of Figures 4.5 and 4.7) of the 20 runs for each packet loss rate. These aggregated results are shown in Figure 4.10.

The boxplots in Figure 4.10 show the spread of the averages, according to the boxplot strategy described in Section 4.9.1.1.

In Tables 4.3,4.4, we can see that with either network implementation, topology consistency and drift distance increases with increasing packet loss.

We can see that UDP outperforms TCP for more than 5% packet loss (99.93% versus 99.83%) and remains very topology consistent (99.51%), even up to 20% packet loss.

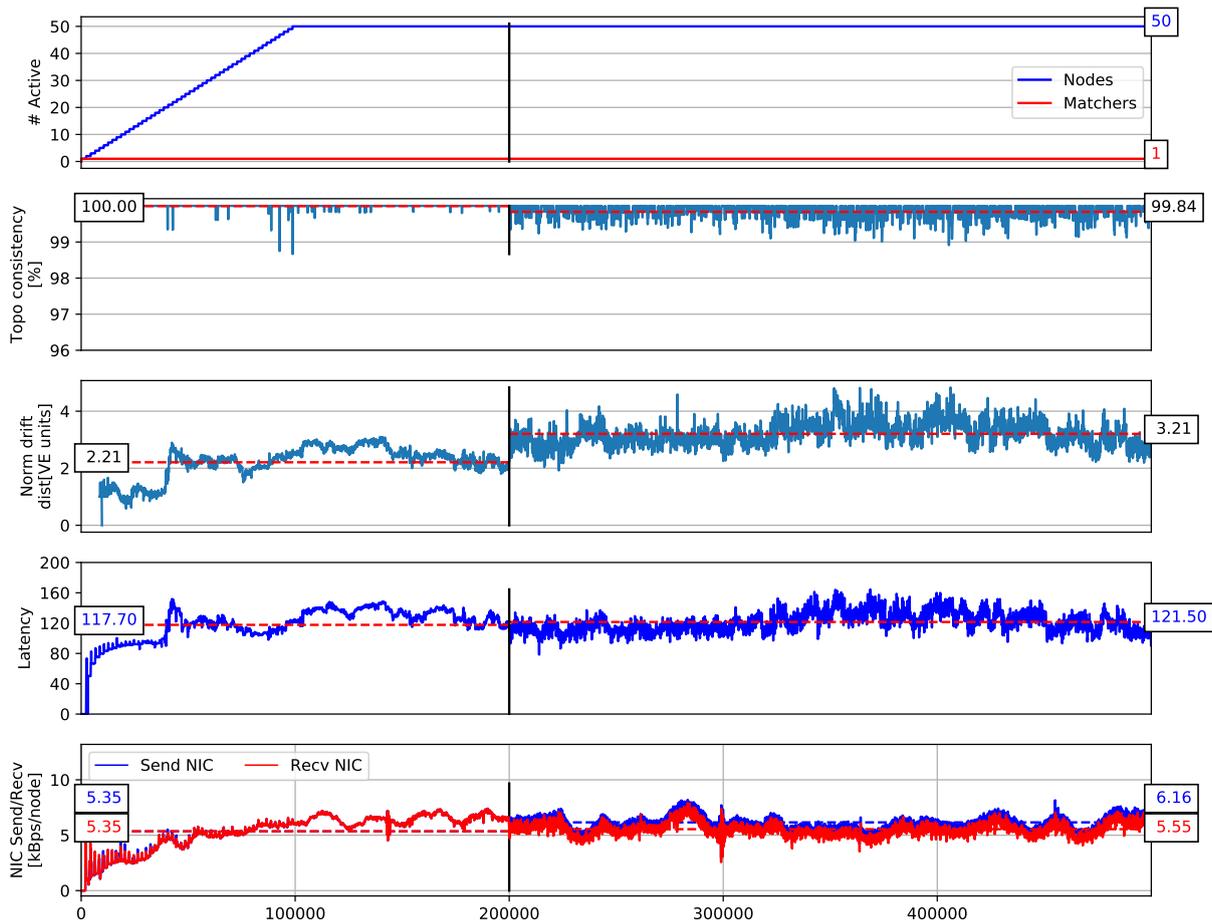


Figure 4.7: Results of a UDP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. In this figure the drift distance plot is scaled to the range of normalised drift distance for readability. See Figure 4.8 for full range of maximum drift distance. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right.

CHAPTER 4. RESULTS OF VAST MMVE UNDER ADVERSE NETWORK
70 CONDITIONS

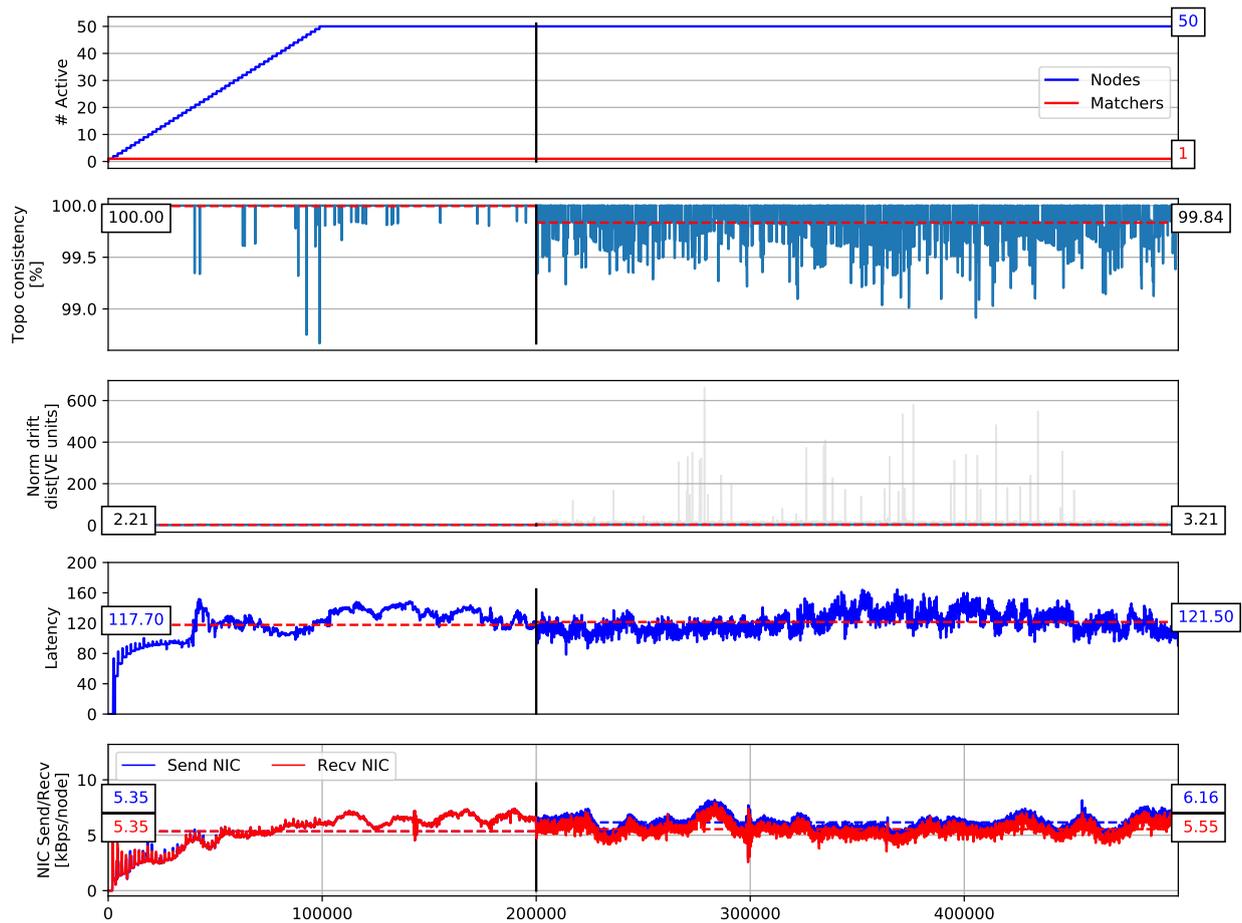


Figure 4.8: Results of a UDP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Topology consistency and drift distance plot shows the full range. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right.

Table 4.4: Medians of normalized drift distances [VE units] with 90% confidence interval, for increasing packet loss test.

	TCP	UDP	± %
0.0 %	2.14 ± 0.00	2.72 ± 0.01	26.99 %
1.0 %	2.62 ± 0.01	2.75 ± 0.01	5.11 %
2.0 %	2.67 ± 0.01	2.71 ± 0.01	1.35 %
5.0 %	3.67 ± 0.01	2.87 ± 0.01	-21.89 %
10.0 %	6.22 ± 0.00	3.21 ± 0.00	-48.39 %
15.0 %	11.51 ± 0.66	3.74 ± 0.01	-67.51 %
20.0 %	55.82 ± 0.59	4.06 ± 0.01	-92.72 %
30.0 %	204.99 ± 0.51	5.78 ± 0.01	-97.18 %
40.0 %	230.28 ± 0.56	8.79 ± 0.01	-96.18 %
50.0 %	245.95 ± 0.42	14.73 ± 0.01	-94.01 %
60.0 %	251.79 ± 0.70	26.64 ± 0.04	-89.42 %
70.0 %	246.36 ± 0.48	55.12 ± 0.15	-77.62 %

Table 4.5: Medians of normalized drift distances [VE units] with degradation percentages, for increasing packet loss test.

	TCP	degrade %	UDP	degrade %
0.0 %	2.14	0.00 %	2.72	0.00 %
1.0 %	2.62	22.05 %	2.75	1.02 %
2.0 %	2.67	24.68 %	2.71	-0.50 %
5.0 %	3.67	71.28 %	2.87	5.35 %
10.0 %	6.22	190.29 %	3.21	17.97 %
15.0 %	11.51	436.54 %	3.74	37.29 %
20.0 %	55.82	2503.00 %	4.06	49.14 %
30.0 %	204.99	9459.70 %	5.78	112.34 %
40.0 %	230.28	10639.08 %	8.79	222.67 %
50.0 %	245.95	11369.52 %	14.73	440.84 %
60.0 %	251.79	11641.89 %	26.64	878.23 %
70.0 %	246.36	11388.67 %	55.12	1924.35 %

Table 4.6: Median of average latencies [ms] with 90% confidence interval, for increasing packet loss test.

	TCP	UDP	± %
0.0 %	118.81 ± 0.17	142.91 ± 0.20	20.29 %
1.0 %	125.48 ± 0.25	141.31 ± 0.41	12.61 %
2.0 %	122.56 ± 0.31	137.70 ± 0.47	12.36 %
5.0 %	129.18 ± 0.31	136.90 ± 0.38	5.98 %
10.0 %	147.44 ± 0.51	132.34 ± 0.18	-10.24 %
15.0 %	180.60 ± 0.57	140.47 ± 0.28	-22.22 %
20.0 %	535.74 ± 4.21	130.89 ± 0.23	-75.57 %
30.0 %	1180.80 ± 6.68	131.21 ± 0.31	-88.89 %
40.0 %	713.32 ± 4.22	126.99 ± 0.19	-82.20 %
50.0 %	432.34 ± 2.75	128.15 ± 0.27	-70.36 %
60.0 %	272.76 ± 1.90	121.14 ± 0.25	-55.59 %
70.0 %	176.50 ± 0.96	199.98 ± 2.10	13.30 %

From Table 4.4 we can clearly see that TCP is sensitive to packet loss, with the drift distance increasing rapidly as more packet loss is applied. We consider more than 30% packet loss the breaking point of TCP: the topology consistency drops rapidly (36.48%) and drift distance spikes (204.99). Although the 30% packet loss result would seem like an anomaly with the 40% packet loss having a better topology consistency, we would argue that the high drift distance in both cases indicates a complete failure of communications. Figure 4.9 shows how the nodes lose connection to the matcher at packet loss rates of 30% and more. No such anomalies were detected in the UDP runs.

Table 4.5 shows the degradation percentages, that is the amount of degradation in terms of drift distance relative to the no packet loss case. TCP degradation (11388%) is much larger than UDP degradation (1924%).

Table 4.6 shows that TCP latency is sensitive to packet loss (eg. 535 ms at 20% packet loss). In contrast, UDP latency remains between 121 - 143 ms for most packet loss percentages, except for 200 ms at 70% loss. However, as the VE state is already degraded at 70% loss, this could be an anomaly.

In Figure 4.10 we can see topology consistency, drift distance, latency as well as the NIC send and receive bandwidth. The boxplots are drawn from the aggregated means of 20 runs per packet loss percentage point. For clarity, only results from 0 - 10% packet loss are shown here. The reasoning for this is TCP degrades very quickly after 10% packet loss, and therefore the y-axis scaling required diminishes resolution of the UDP result. 1% packet loss results were

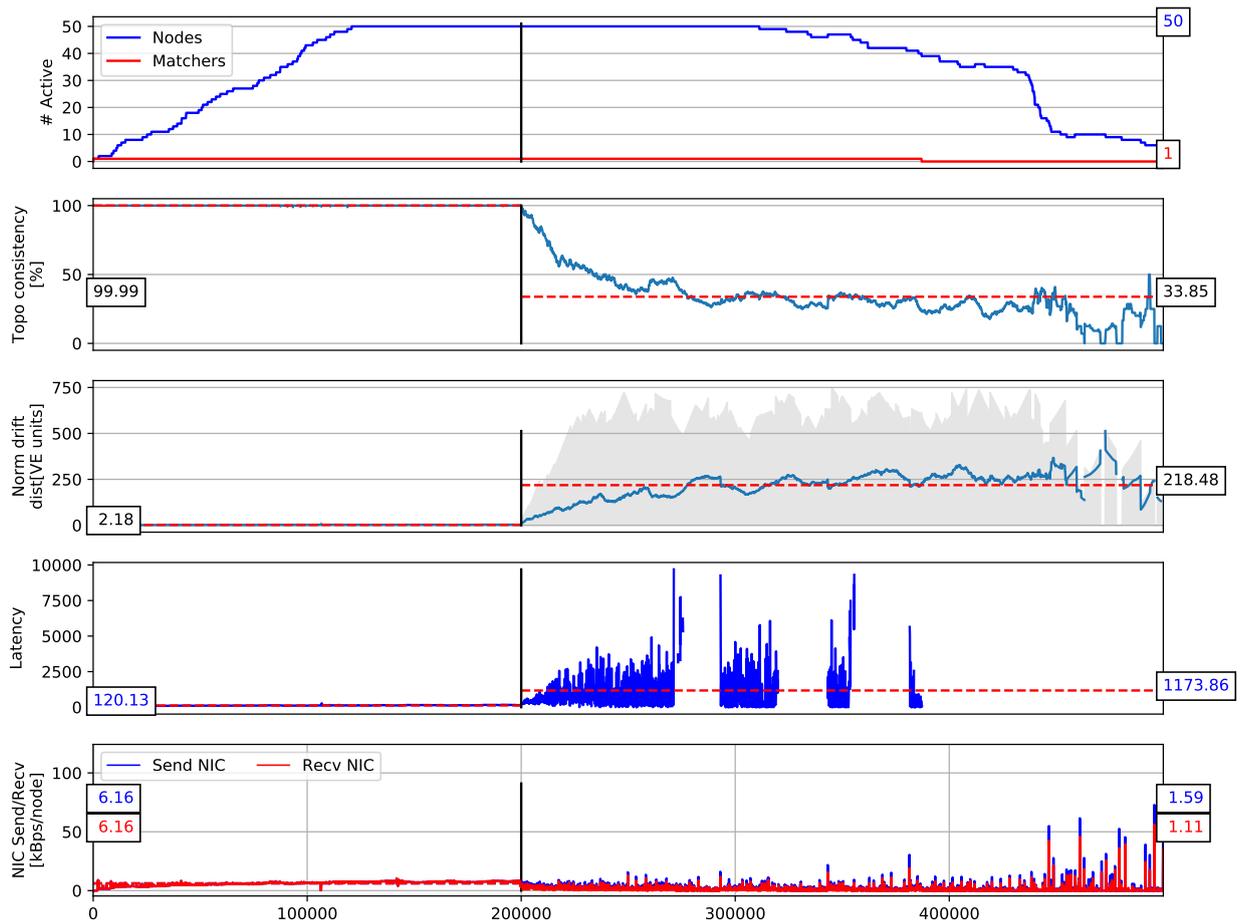


Figure 4.9: Results of a 30% packet loss TCP run, showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Topology consistency and drift distance plot shows the full range. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right.

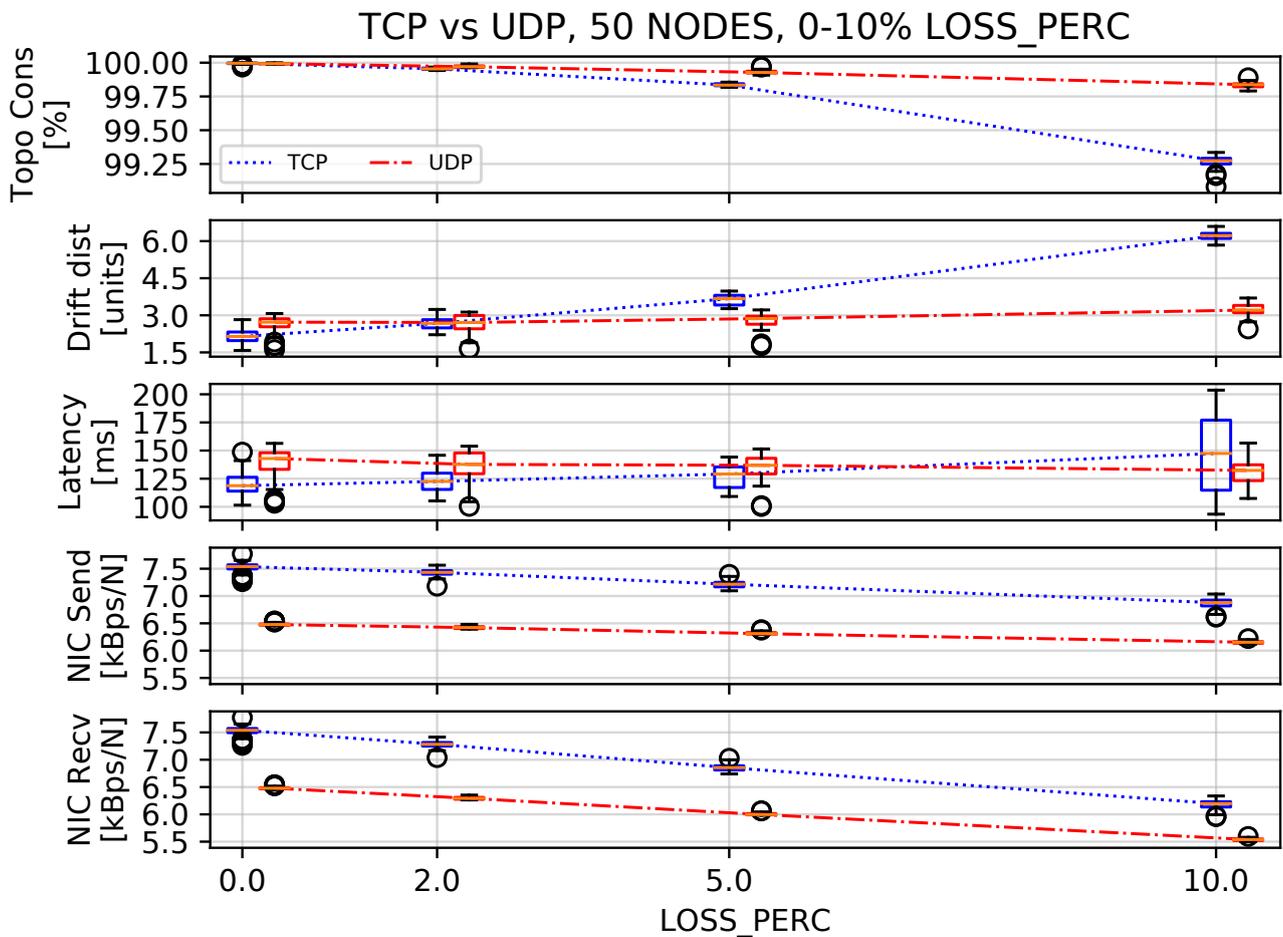


Figure 4.10: Aggregated summary of 20 runs, each with 50 nodes. TCP results shown in blue and UDP in red. Outliers are indicated with black circles. Only packet loss percentages up to 10% is shown in this graph to facilitate readability. Full results are shown in Tables 4.3,4.4.

also omitted so that trends can be seen more clearly.

Latency of TCP increases as packet loss increases in contrast to UDP latency decreasing as packet loss increases.

In general, UDP sends and receives less than TCP. It can also be observed that both UDP and TCP sends and receives less bytes as packet loss increases. This can be explained by the packet loss influencing the interactivity of the application, explained in the discussion below.

4.11.6 Discussion

It is clear from the above results that UDP is less sensitive to packet loss than TCP. The main reason for this is the reliability requirement of TCP competing with increasing packet loss. As more packet loss is applied, more TCP retransmissions are needed to recover the missing packets in the stream. However, these retransmissions are also subject to packet loss, leading to a compounding of the packet loss effect. Furthermore, TCP acknowledgments are also subject to packet loss, leading to erroneous retransmissions as well. All the retransmissions has a detrimental effect on latency. This in turn has a strong effect on drift distance and finally an effect on topology consistency. In the extreme case this packet loss induced latency increases to the point where the client does not receive an update for a couple of steps and assumes that it is disconnected, leading to a total failure of the VE.

There is a notable trend in message latency data: UDP latency decreases with packet loss, and TCP latency increases with packet loss. TCP latency has already been explained by retransmissions. However, UDP latency results are unexpected as packet loss is not expected to influence UDP latency. However, this latency decrease can be explained by the single-threaded nature of UDP: because less packets are received under packet loss conditions, less packets need to be processed. Only latency of packets successfully received are processed, leading to lower total processing time overall - those packets that *do* arrive are processed immediately.

In terms of Quality of Experience (QoE), the drift distance induced by packet loss can be expressed as a lag in terms of game steps or even experienced latency. Note that the experience latency is different from the actual application measured latency as the latter is based on the time it took for successful transmission whereas the former is based on when the user experiences a response to their actions. Clients move around the VE at a speed of 3 VE units per timestep. Therefore, each multiple of 3 VE units drift distance can be described as one timestep lag. Stated differently, each multiple of 3

VE can be expressed as a QoE degradation of 100 ms. For example, the increase in drift distance for TCP between 0% and 10% packet loss (4.08 VE units) can be described as 1.36 timesteps of QoE lag or roughly 136 ms lag. Although this is not an accurate comparison as the VE application also influences QoE, it does provide a broad-strokes description of degradation in terms of user experience.

An unexpected decline in NIC send and received bandwidth was observed for both network implementations. We describe this as “loss of interactivity” and can be explained by how VAST handles neighbour updates. Typically, the VAST Matcher notifies the new neighbours of a node of its presence, so that these neighbours can add it to their neighbour lists. However, if the initial neighbour notification packet was lost, and a client receives an update from a neighbour that it has not seen before, it will request the neighbour’s full information, such as AOI, ID and neighbours. In this case the client is aware of what is information it is missing, and can request it. Under higher packet loss conditions, the initial neighbour message and subsequent state updates can be lost, causing the client to not issue such a request. The client is unaware of what it is missing, and cannot respond accordingly. In the extreme case, all nodes are oblivious of each other and interactivity is completely lost. This loss of interactivity is reflected in the downward trend in NIC send bytes. As interactivity degrades, client stop requesting extra information, leading to a lower send bandwidth. The slope of decline is less for UDP than TCP, indicating that although UDP interactivity is more robust against loss, with both network implementations state consistency decreases as packet loss increases.

4.11.7 Why does TCP Perform Poorly?

A number of possible reasons why TCP performs poorly can be found in literature:

4.11.7.1 TCP Mistakes Packet Loss for Congestion

According to Fu et al [60], TCP detects packet drops as a congestion signal, as packet loss in wired networks are caused by buffer overflows. Buffer overflows in wired networks indicate that the offered load to the network exceeded the capacity at one of the intermediate routers, which cannot keep up, and therefore drops packets as the buffer overflows. In response to a congestion signal, TCP reduces the Maximum Segment Size, thereby reducing the bandwidth requirements. However, in wireless networks, packet drops occur due to the hidden node problem and channel contention. In our

setup, packet losses are emulated probabilistically and also do not occur due to buffer overflows but rather independently of the offered network load (similar to wireless networks). TCP therefore performs poorly as it wrongly interprets packet drops as a congestion signal, unnecessarily throttling communication.

4.11.7.2 Virtual Environment Traffic Patterns

Typical traffic patterns of an interactive VE is not well suited to TCP communication. According to Chen et al. [45], VE traffic has four characteristics that may interfere with TCP communication: state updates are delivered in very small packets, traffic generation is application limited, packet rates are low, and bi-directional traffic is frequent. Firstly, small packets incur large header overheads (up to 46% of total bandwidth in Chen's analysis).

Secondly, application-limited traffic generation interferes with TCP congestion control; when the connection has been idle for too long, the window size is reset to minimum in order to avoid congestion under unknown network conditions.

Finally, the low packet rate and bi-directional traffic prevent effective use of the fast-retransmit in TCP. Because the packet rates are low, too few packets are sent and acknowledged for the three duplicate ACK retransmit to be triggered. Furthermore, if traffic is sent from the other side (i.e. bi-directional flow), it is counted as a successful transmission, and the three duplicate ACK counter is reset, also preventing the use of fast-retransmit. Due to these two effects, 99% of the packet loss recovery occurs through retransmit timeouts instead of fast-retransmit, which add delay (on average 700 ms in Chen's study) for packets lost.

Chen gives the following recommendations for VE protocol designers: allow reliable and unreliable transmission, ordered and unordered transmission, parallel TCP streams for unrelated messages and coordinated congestion control. Relaxing reliability and ordered requirements for some messages (based on their relevance in the VE) can reduce the network load without significantly reducing the VE experience. Parallel TCP streams can decouple the delays for unrelated messages in the same TCP queue. In coordinated congestion control, clients and servers of the VE can communicate their expected bandwidth requirements and ensure fairness in bandwidth use on time scales larger than typical TCP transfer windows. Coordination of transmissions can also reduce burstiness in communication.

Griwodz and Halvorsen [64] agree with Chen's results and suggest that a more aggressive retransmission scheme, for example a

retransmission after every duplicate ACK, could reduce the latency associated with packet loss.

Petlund et al. [98] also suggest a more aggressive retransmission scheme, as well as removing exponential back-off and adding data bundling. Low packet rates and therefore long inter-arrival times of packets limit the TCP transfer window increase opportunities. Therefore removing exponential back-off on packet loss would reduce retransmission delay; otherwise the transfer window would only recover after many more successfully received packets.

The suggestions in [64] and [98] could improve on the TCP results of this test. However, due to time constraints this is considered future work.

4.11.8 Comparison with Earlier Work

One of the purposes of this project was to analyse the effect of packet loss on a VE in a comparable way. However, due to processing power limitations of our setup, we could not run as many nodes as was present in other papers written by Hu and others. However, as far as possible we will attempt to relate similar situations in their previous papers to our tests. We list and discuss them here:

- In *VON: A scalable peer-to-peer network for virtual environments* [69], Hu, Chen and Chen present a test of 200 nodes, 5 units / step movement speed and 100 units AOI. The resulting average transmission bandwidth was 2.5 kBps / node under no packet loss. This can be related to our 5.07 kBps / node. Possible reasons for differences include that our test was run at a slower movement rate (reducing bandwidth), but with a larger AOI (increasing visible neighbours, increasing bandwidth). It is encouraging to see that the results are in the same order of magnitude.

They also perform a packet loss test, showing topology consistency dropping as packet loss increases. Although it follows a similar pattern to our tests, we experienced complete consistency loss at much lower packet loss percentages. We propound that the discrepancy can be explained by the difference between our networking setup and packet loss application, compared to Hu's: although it is unclear what testbed was used in their test, packet loss was applied on the application level and control messages were exempt from packet loss. Our test involved a full network stack with packet loss applied at the network level, without exceptions. Processing time in

a full network stack, agnostic packet loss and possibly TCP's excessive retransmissions could explain the discrepancy.

- In *A spatial publish subscribe overlay for massively multiuser virtual environments* [71], Hu et al presents a average latency test for different numbers of relays. In our tests we allow each node to be its own relay, similar to their 90 relay, 90 nodes test. Their value of around 120 ms latency corresponds well to our 118.8 ms.

4.11.9 Summary

In the packet loss test we showed that TCP is very sensitive to packet loss. Although UDP would be a more effective transport protocol in these scenarios, it is still subject to packet loss induced state consistency degradation. In general, packet loss decreases topology consistency, increases drift distance and results in a loss of interactivity.

We also compared our test results to previous papers' results and found evidence that our results are comparable, indicating that our system performs as expected even with a full network stack.

4.12 Comparing Mininet Hosts with Physical Hosts

In order to verify our results in the packet loss test from the previous section, we evaluate the performance of VAST on a system with a OpenFlow switch and physical hosts.

However, the OpenFlow switch that could be obtained was a PC running the Open Virtual Switch [12] software and only allowed four physical hosts to connect. Therefore we evaluate the performance using four physical hosts. For comparison we also evaluated the performance in Mininet using four Mininet hosts.

4.12.1 Experiment Setup

In order to obtain the Mininet results, we follow the exact same procedure as was specified in Section 4.11.1, except we only use four Mininet hosts instead of 50.

In every aspect possible the physical hosts are identical to the Mininet hosts:

- The same VAST library software is also used on both physical hosts and Mininet hosts.

CHAPTER 4. RESULTS OF VAST MMVE UNDER ADVERSE NETWORK
80 CONDITIONS

Table 4.7: Mean and standard deviation of 100 round-trip ping times on Mininet and physical network. Results in microseconds.

Link	Mininet		Physical	
	Mean	Std dev	Mean	Std dev
h1 -> h2	43.8	3.89	253.3	37.8
h2 -> h1	42.2	4.00	260.8	35.2

Table 4.8: Comparison test simulation parameters

Packet loss	0-20%
Number of nodes	4

- The same Linux operating system is used on both physical and Mininet hosts.
- The same POX controller as is used with the Mininet setup.

Therefore the only differences between the physical hosts are the following:

- Physical wires are used to connect to the switch, which could cause a small delay. The round-trip latency was measured and the results are given in Table 4.7.
- The host clocks are synchronised with the Network Time Protocol (NTP).
- Each of the clients run on a separate host and therefore the processing delay would not influence each other. This would be beneficial to the state consistency as processing delay per host is reduced.
- Packet loss rates are applied using the Linux `iptables` [6] filters at the receiving hosts, instead of by Mininet (using `tc`) on network links. `iptables` controls the packet filtering in the Linux kernel, whereas `tc` controls the Linux network scheduler in the kernel. Both utilities allow the probabilistic dropping of packets. However, as `tc` only manipulates outgoing packets and we wanted to drop packets on the downstream link, we decided to use `iptables` to apply packet loss at the receiving hosts. Applying packet loss at the switch was not possible as we did not have control over the internal functioning of OVS.

In addition to the simulation parameters given in Table 4.1, this specific test uses the simulation parameters given in Table 4.8.

Table 4.9: Median of normalised drift distances [VE units] and 90% confidence intervals for TCP test when comparing Mininet and physical hosts.

	Mininet	Physical	\pm %
0.0 %	2.05 \pm 0.01	1.77 \pm 0.01	-13.62 %
10.0 %	4.48 \pm 0.01	5.22 \pm 0.01	16.39 %
20.0 %	18.18 \pm 0.06	22.56 \pm 0.29	24.06 %

Table 4.10: Median of average latencies [ms] and 90% confidence intervals for TCP test when comparing Mininet and physical hosts.

	Mininet	Physical	\pm %
0.0 %	93.63 \pm 0.38	108.65 \pm 0.41	16.04 %
10.0 %	126.29 \pm 0.20	135.28 \pm 0.20	7.12 %
20.0 %	235.73 \pm 0.30	263.62 \pm 0.43	11.83 %

Table 4.11: Median of normalised drift distances [VE units] and 90% confidence intervals for UDP test when comparing Mininet and physical hosts.

	Mininet	Physical	\pm %
0.0 %	2.07 \pm 0.01	1.60 \pm 0.01	-22.82 %
10.0 %	2.59 \pm 0.01	1.91 \pm 0.01	-26.37 %
20.0 %	2.88 \pm 0.01	2.78 \pm 0.01	-3.47 %

4.12.2 Results

From Table 4.10 and Table 4.12 it can be observed that the physical network generally has a higher latency.

Figure 4.11 and Figure 4.12 show the comparison between performance using Mininet hosts and physical hosts. From both figures it can be seen that the emulated and physical network perform similarly.

The most prominent difference between the Mininet and physical networks are present in the NIC receive graphs. Unlike Mininet where received bandwidth steadily decreases as the packet loss rate increases, the NIC receive bandwidth remains equal to NIC send bandwidth for this physical network setup. This can be explained

Table 4.12: Median of average latencies [ms] and 90% confidence intervals for UDP test when comparing Mininet and physical hosts.

	Mininet	Physical	\pm %
0.0 %	98.16 \pm 0.32	93.86 \pm 0.42	-4.38 %
10.0 %	97.95 \pm 0.38	88.87 \pm 0.58	-9.27 %
20.0 %	93.00 \pm 0.25	113.86 \pm 0.59	22.43 %

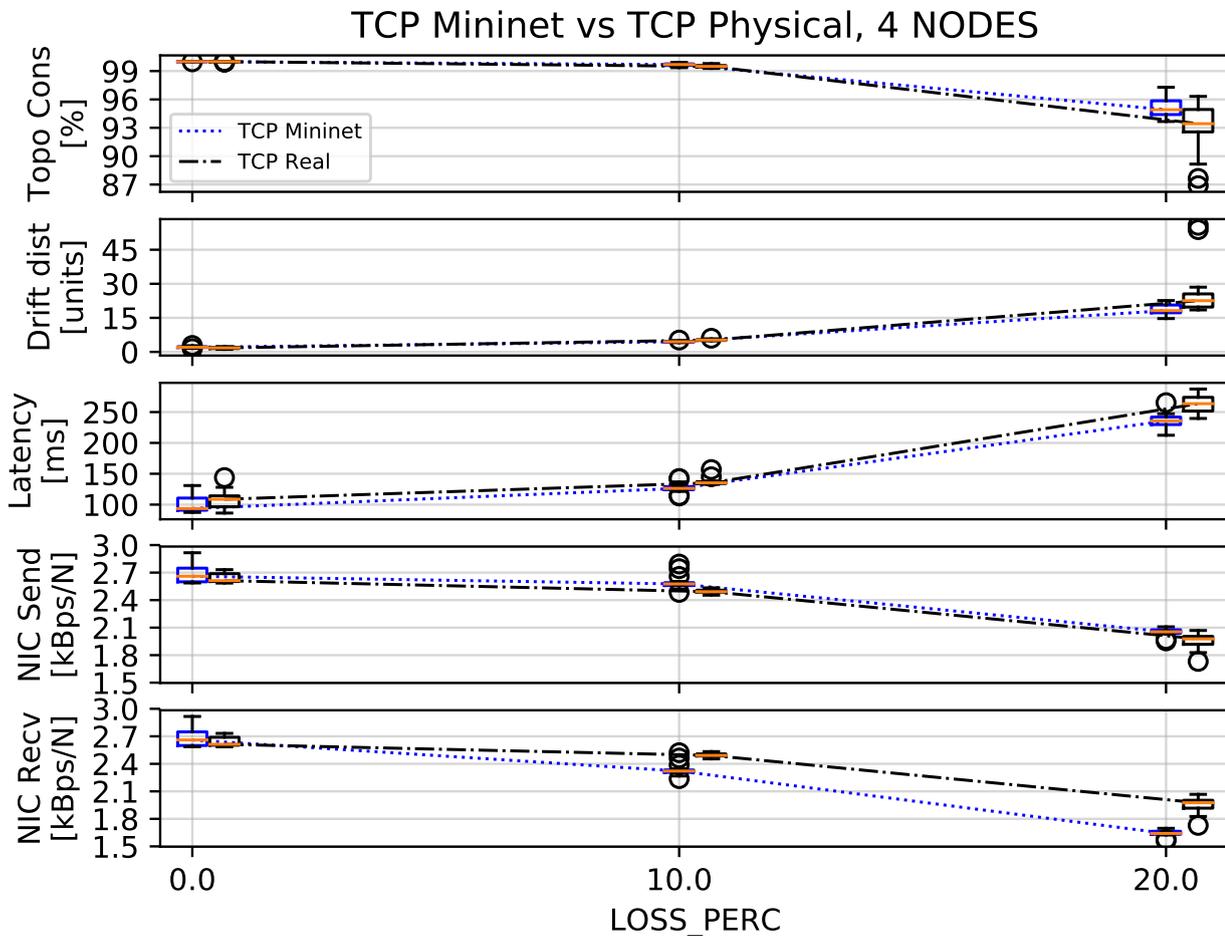


Figure 4.11: Mininet hosts versus physical hosts using TCP. The Mininet performance is shown in blue and the physical network performance is shown in black. Outliers are indicated as black circles.

by the way in which packet loss is applied on the physical hosts. In both network setups, the NIC bandwidth is measured as the bytes that enter the interface. However, in the Mininet network, packet loss is applied between the switch and the receiving host, while on the physical host packet loss is applied in the filtering tables, after the bytes have been received. From the application perspective the same loss is experienced, and therefore the adverse network conditions are the same. However, from the NIC perspective all of the bytes are still received and then dropped later in the pipeline, leading to the incorrect NIC bandwidth measurement. In physical network with physical packet loss in transmission, the NIC receive bandwidth would also reflect the loss as the Mininet network does.

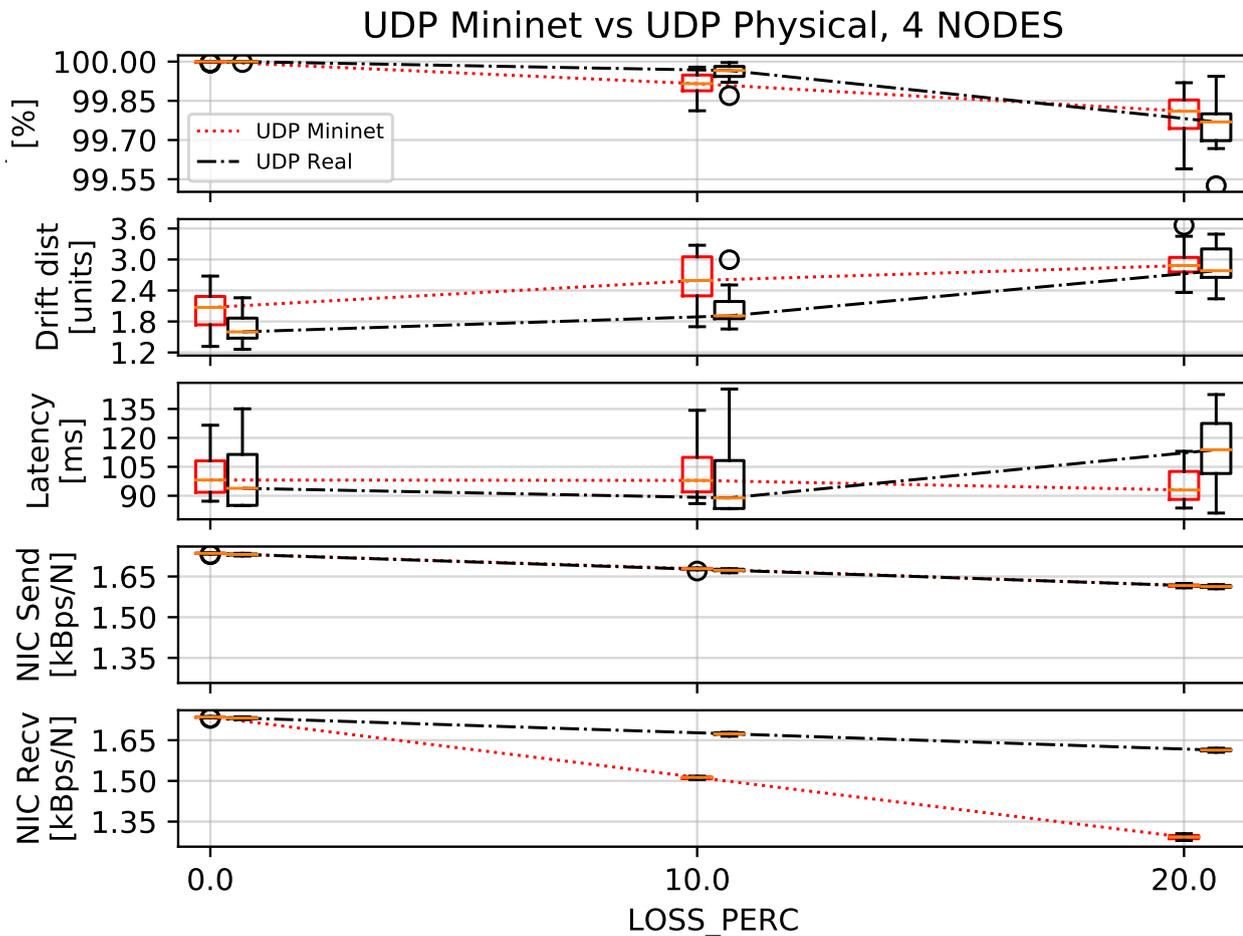


Figure 4.12: Mininet hosts versus physical hosts using UDP. The Mininet performance is shown in red and the physical network performance is shown in black. Outliers are indicated as black circles.

4.12.3 Discussion

With both TCP and UDP tests, latency for the physical network is higher than that of Mininet. This can be explained by the physical wires and OpenFlow switch delaying transmission slightly.

In accordance with the higher latency, TCP drift distance is slightly higher (5.23 instead of 4.48 at 10% packet loss) for the physical network test compared to the Mininet test.

The improved drift distance of the UDP physical hosts test can possibly be explained by the reduced processing delay of UDP sockets running on completely separate physical hosts.

At 20% packet loss, the performance of the UDP test using phys-

ical hosts is roughly the same as the Mininet test. It is difficult to determine exactly why the performance of UDP on physical hosts degrades faster than Mininet hosts, but we suspect that the packet delay variation (jitter) is more prominent on physical hosts and could therefore have a detrimental effect in terms of latency and drift distance.

4.12.4 Summary

The differences in latency and state consistency between the Mininet emulated network and the physical network can be explained by the increased network delay and jitter of physical hardware. Despite these small differences, the comparison verifies that the performance of VAST using the Mininet emulated network corresponds closely with the performance using a real physical network.

Ideally all further testing in this chapter should be performed using the physical hosts. However, due to hardware and time constraints, the rest of the tests will be performed using Mininet.

4.13 Delay Tests

Network delay is another adverse network conditions that could affect state consistency. We use Mininet to apply transmission delay to our network links and observe the effects. Unfortunately Mininet does not model network delay variation. Network delay variation would be possible to measure in a real network scenario and is therefore considered future work.

4.13.1 Experiment Setup

We once again use a custom Mininet topology to specify network delay. Tests are run in the same way as specified in section 4.9.1. Like the packet loss tests above, we only applied the delay after the simulation setup steps (see section 4.8). It should be noted that when we refer to network delay, it is the delay of a packet sent from one network interface to another, without the processing delay. The latency result refers to the application-measured latency experienced by VAST messages, which includes the network delay, processing delay and enqueued delay while the packet waits to be processed in the next update step.

In addition to the simulation parameters given in Table 4.1, this specific test uses the simulation parameters given in Table 4.13. A

Table 4.13: Delay test simulation parameters

Packet loss	10%
Number of nodes	50
Delay	0 - 200 ms

Table 4.14: Medians of normalized drift distances [VE units] with 90% confidence interval when adding network delay.

	TCP	UDP	± %
0.0 ms	6.23 ± 0.00	3.22 ± 0.00	-48.36 %
20.0 ms	9.69 ± 0.00	4.56 ± 0.01	-52.93 %
50.0 ms	17.50 ± 0.01	4.78 ± 0.00	-72.66 %
100.0 ms	25.93 ± 0.01	8.94 ± 0.01	-65.54 %
200.0 ms	41.94 ± 0.01	14.95 ± 0.01	-64.36 %

Table 4.15: Median of normalized drift distances [VE units] with degradation percentages when adding network delay.

	TCP	degrade %	UDP	degrade %
0.0 ms	6.22	0.00 %	3.21	0.00 %
20.0 ms	9.69	55.73 %	4.56	42.03 %
50.0 ms	17.5	181.11 %	4.78	48.95 %
100.0 ms	25.93	316.56 %	8.94	178.17 %
200.0 ms	41.94	573.75 %	14.95	365.26 %

constant packet loss value of 10% was chosen to represent lossy wireless network conditions.

4.13.2 Results

From Table 4.14 we can see that drift distance degrades quickly for TCP (eg. 17.50 at 50 ms) under increased network delay. UDP drift distance degrades at a slower rate (eg. only 4.78 at at 50 ms).

From Table 4.16 is clear that there is no linear relationship be-

Table 4.16: Median of average latencies [ms] with 90% confidence interval when adding network delay.

	TCP	UDP	± %
0.0 ms	130.71 ± 0.41	130.52 ± 0.21	-0.14 %
20.0 ms	227.55 ± 0.20	187.74 ± 0.50	-17.49 %
50.0 ms	217.98 ± 0.03	177.54 ± 0.08	-18.55 %
100.0 ms	188.84 ± 0.02	298.62 ± 0.26	58.14 %
200.0 ms	878.28 ± 0.13	525.71 ± 0.44	-40.14 %

Table 4.17: Median of normalised NIC send bandwidth [kBps/node], 90% confidence intervals and total bandwidth of the network when adding network delay.

	TCP	total	UDP	total	± %
0.0 ms	6.88 ± 0.00	343.92	6.15 ± 0.00	307.65	-10.54 %
20.0 ms	7.01 ± 0.00	350.4	6.22 ± 0.00	311.23	-11.18 %
50.0 ms	5.59 ± 0.00	279.27	6.23 ± 0.00	311.59	11.57 %
100.0 ms	3.94 ± 0.00	196.85	6.13 ± 0.00	306.67	55.79 %
200.0 ms	2.79 ± 0.00	139.44	6.14 ± 0.00	306.82	120.04 %

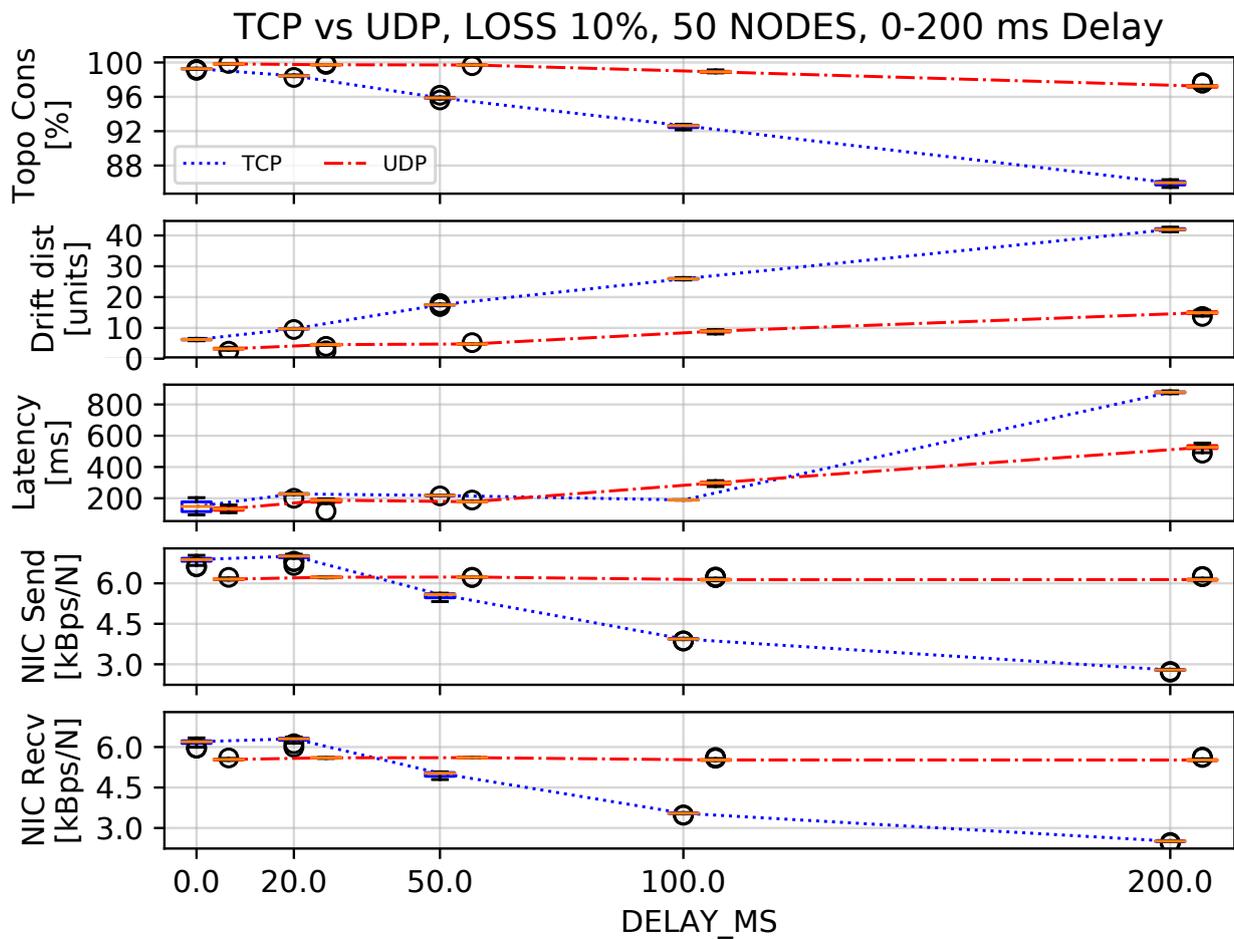


Figure 4.13: Network delay of 0 - 200 ms. TCP results shown in blue and UDP in red. Outliers are indicated with black circles.

tween network delay and experienced latency. It is clear that a network delay of more than one step duration seriously increases experienced latency of both TCP and UDP (eg. 878.28 and 525.71 ms respectively at 200 ms) as well as greatly increasing drift distance (41.94 and 14.95 respectively).

Table 4.15 shows the degradation percentages due to network delay compared to the no network delay case. At 200 ms, a degradation of 573 % and 365 % in drift distance for TCP and UDP respectively can be observed.

Table 4.17 and Figure 4.13 shows NIC send and receive bandwidth decreases as network delay increases, showing that similar to packet loss, latency degrades the interactivity of the application.

4.13.3 Discussion

TCP reliable transmission requires retransmits when a segment is not acknowledged in due time: the TCP sender sets a timeout (typically based on the estimated round-trip time) for acknowledgment to arrive. If not acknowledged within the timeout period, TCP retransmits the packet. With each missed acknowledgment, the timeout is increased (typically doubled).

In network delay conditions, either the flow control strategy decreases the transfer rate to avoid late acknowledgments or timeout retransmissions cause congestion in the network. The NIC send bandwidth indicates that the former is correct as bandwidth does not increase with latency, but rather decreases.

4.13.4 Summary

Similar to packet loss, network delay also adversely affects state consistency, typically resulting in higher experienced latency than the network delay itself. The increased latency and drift distance will be experienced by the VE users as lag and will degrade the VE experience.

4.14 Bandwidth Limited Tests

4.14.1 Experiment Setup

The custom Mininet topology in this case specifies the bandwidth limit. Tests are run in the same way as specified in section 4.9.1. We applied the bandwidth limitation after the simulation setup steps.

Table 4.18: Bandwidth limit simulation parameters

Packet loss	0%
Number of nodes	50
Bandwidth limits	0.2 - 5 MBps

Table 4.19: Median of normalised drift distances [VE units] and 90% confidence interval when limiting transmission bandwidth.

	TCP	UDP	\pm %
5.0 MBps	2.59 \pm 0.01	2.74 \pm 0.01	5.78 %
2.0 MBps	2.70 \pm 0.01	2.80 \pm 0.01	3.80 %
1.0 MBps	2.69 \pm 0.01	2.89 \pm 0.01	7.40 %
0.5 MBps	4.67 \pm 0.01	2.84 \pm 0.01	-39.27 %
0.4 MBps	9.22 \pm 0.01	3.00 \pm 0.01	-67.51 %
0.2 MBps	257.80 \pm 0.13	121.58 \pm 0.12	-52.84 %

In addition to the simulation parameters given in Table 4.1, this test uses the simulation parameters given in Table 4.18. Unlike the delay test, we decided not to apply packet loss. This is because the bandwidth limitation has a very sharp decline in performance after 0.5 MBps and adding packet loss would result in a complete failure of the VAST overlay.

The bandwidth limit is applied on each link. The bandwidth requirements in this project is low, as was explained in section 4.4. However, the matcher's links carries the most bandwidth in the network as it needs to receive all of the publications and distribute them to the correct subscribers. Thus even though the bandwidth available in our intended application (local direct-connected VE experience) likely exceeds the expected bandwidth requirement, we perform this test to investigate what would happen in a bandwidth constrained network.

4.14.2 Results

Table 4.19 shows that nominal performance is observed for most bandwidth limitations until 0.4 MBps. State consistency deterioration occurs for TCP at 0.5 MBps with drift distance increasing from 2.69 to 4.67. A complete loss of consistency occurs at 0.2 MBps with a drift distance of 257.80 VE units. A similar failure occurs for UDP at 0.2 MBps with drift distance increasing from 3.0 to 121.58.

Table 4.20 shows the degradation percentages of the drift distance. It can be seen that both TCP and UDP suffers severe degradation (9841 % and 4332 %) at the 0.2 MBps bandwidth limit.

Table 4.20: Median of normalised drift distances [VE units] with degradation percentages when limiting transmission bandwidth.

	TCP	degrade %	UDP	degrade %
5.0 MBps	2.59	0.00 %	2.74	0.00 %
2.0 MBps	2.7	3.94 %	2.8	2.00 %
1.0 MBps	2.69	3.79 %	2.89	5.38 %
0.5 MBps	4.67	80.23 %	2.84	3.47 %
0.4 MBps	9.22	255.62 %	3	9.23 %
0.2 MBps	257.8	9841.11 %	121.58	4332.29 %

Table 4.21: Median of average latencies [ms] and 90% confidence interval when limiting transmission bandwidth.

	TCP	UDP	± %
5.0 MBps	140.10 ± 0.39	143.01 ± 0.52	2.08 %
2.0 MBps	143.69 ± 0.33	146.52 ± 0.27	1.97 %
1.0 MBps	142.23 ± 0.28	150.03 ± 0.33	5.48 %
0.5 MBps	217.07 ± 0.22	147.36 ± 0.22	-32.11 %
0.4 MBps	349.87 ± 0.21	153.70 ± 0.25	-56.07 %
0.2 MBps	2982.19 ± 4.02	3711.79 ± 2.70	24.46 %

Table 4.22: Median of normalised NIC receive bandwidth [kBps/node], 90% confidence intervals and total bandwidth of the network when limiting transmission bandwidth.

	TCP	total	UDP	total	± %
5.0 MBps	7.69 ± 0.00	384.66	6.47 ± 0.00	323.43	-15.92 %
2.0 MBps	7.80 ± 0.00	390.14	6.49 ± 0.00	324.31	-16.87 %
1.0 MBps	7.84 ± 0.00	391.77	6.49 ± 0.00	324.7	-17.12 %
0.5 MBps	7.07 ± 0.00	353.32	6.51 ± 0.00	325.38	-7.91 %
0.4 MBps	5.63 ± 0.00	281.52	6.50 ± 0.00	325.05	15.46 %
0.2 MBps	1.17 ± 0.00	58.43	4.18 ± 0.01	209.12	257.87 %

From Table 4.21, it can be seen that at 0.2 MBps the TCP and UDP take long to receive the latest state update messages (roughly 2982 ms or 30 steps for TCP and 3711 ms or 37 steps for UDP).

From Table 4.22 we can see that the total bandwidth of either protocol is less than the bandwidth limitation, as expected. It can be seen that TCP transmission is much more affected with only 58.43 kBps at 0.2MBps limit, much lower than the allowed bandwidth. UDP received bandwidth is 209.1 kBps at 0.2MBps limit, which is roughly the bandwidth limit.

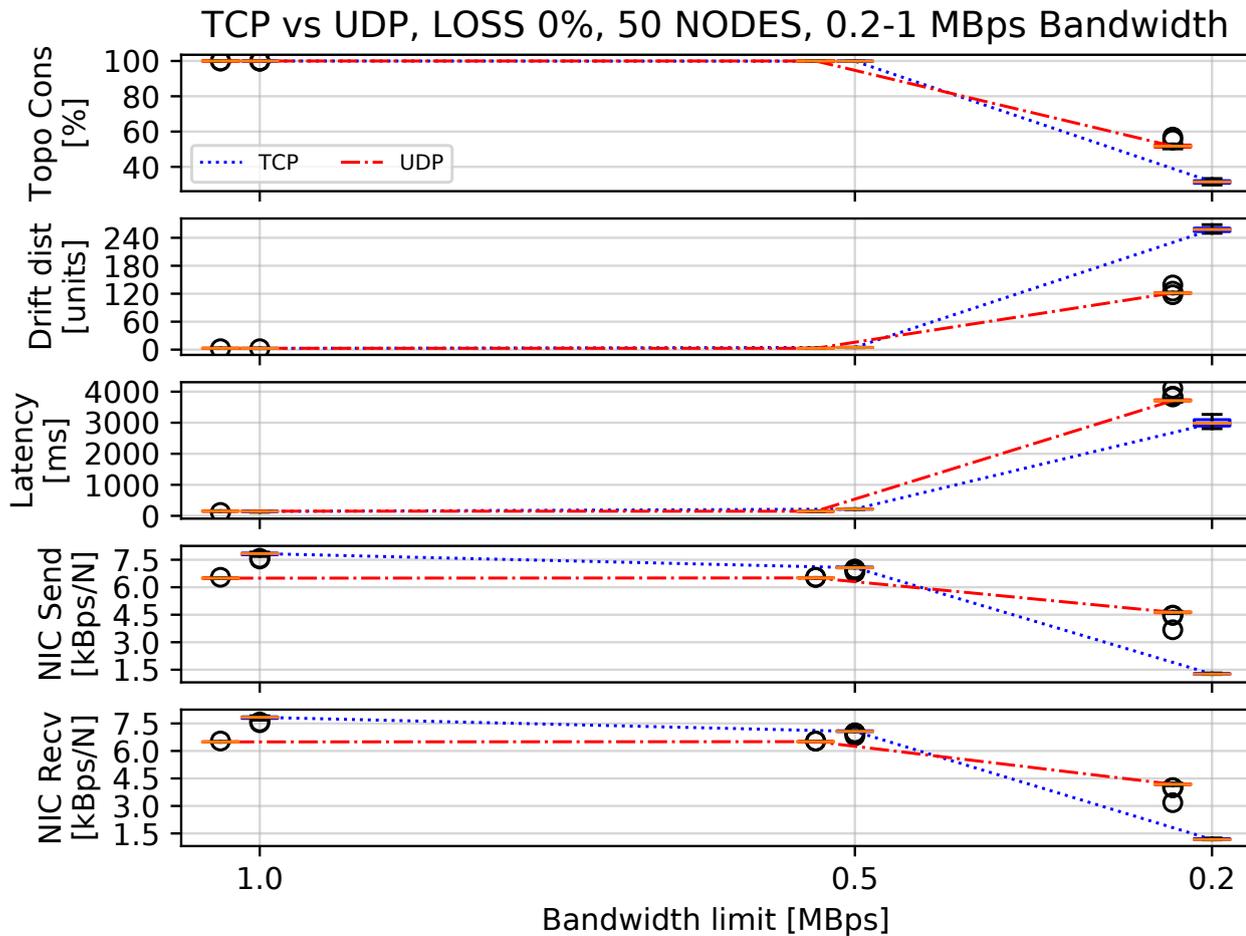


Figure 4.14: Bandwidth limitation tests. The x-axis indicates an increasingly strict limitation on the bandwidth. Outliers are indicated with black circles. Only the range of 1 MBps is shown as nominal state consistency was observed for higher allowed bandwidth. Table 4.21 shows all of the results of the bandwidth test.

Table 4.23: Scalability test simulation parameters

Packet loss	10%
Number of nodes	20 - 100
Total Simulation steps:	
20 - 60 nodes	5000
70 - 100 nodes	10000

4.14.3 Discussion

For both TCP and UDP the state consistency deterioration occurs because packets cannot be sent at the required rate and are delayed by waiting in the NIC send queue. TCP is especially sensitive to the bandwidth limitation as the increased delay causes the flow control algorithm to exponentially reduce the transmission rate. UDP does not perform flow control, therefore queuing packets at the sender increases the experienced latency even beyond that of TCP.

4.14.4 Summary

The rapid increase in drift distance at 0.2 MBps would reduce inter-activity almost completely: users would have to wait 30 steps and 37 steps for TCP and UDP respectively. Therefore a network that can support the bandwidths required to host the VE is vital for successful operation.

4.15 Scalability Tests

In this section we will investigate the effect of scale on the state consistency. Although this is not a test of adverse network effects, our tests will show how scalable a full network stack implementation under packet loss conditions is.

4.15.1 Experiment setup

In order to represent the typical lossy wireless running conditions, we run the scaling tests at 10% packet loss.

We use a single PC for our Mininet simulations which consisted of 16 CPU cores and 64 GB of RAM. However, during simulations, all of the nodes' processing is done on the same physical machine, which used 90% of the CPU power and 12 GB of RAM with 100 nodes. We therefore limit our test to 100 nodes as larger numbers would likely introduce processing artifacts and distort results.

Table 4.24: Median of normalised drift distances [VE units] and 90% confidence interval when increasing the number of nodes.

# Nodes	TCP	UDP	± %
20	5.84 ± 0.01	2.45 ± 0.00	-58.03 %
30	5.96 ± 0.00	2.64 ± 0.01	-55.68 %
40	6.04 ± 0.00	2.95 ± 0.01	-51.20 %
50	6.22 ± 0.00	3.21 ± 0.00	-48.39 %
60	6.02 ± 0.00	3.34 ± 0.01	-44.49 %
70	6.17 ± 0.00	3.59 ± 0.01	-41.79 %
80	5.97 ± 0.00	3.78 ± 0.01	-36.64 %
90	5.87 ± 0.00	3.75 ± 0.01	-36.19 %
100	5.80 ± 0.00	3.94 ± 0.01	-32.01 %

Table 4.25: Median of average latencies [ms] and 90% confidence interval when increasing the number of nodes.

# Nodes	TCP	UDP	± %
20	133.14 ± 0.13	100.19 ± 0.07	-24.75 %
30	133.14 ± 0.14	109.67 ± 0.21	-17.63 %
40	139.51 ± 0.21	124.42 ± 0.31	-10.82 %
50	130.71 ± 0.41	130.52 ± 0.21	-0.14 %
60	153.28 ± 0.25	143.93 ± 0.40	-6.10 %
70	149.40 ± 0.28	151.80 ± 0.22	1.61 %
80	146.44 ± 0.26	159.09 ± 0.22	8.64 %
90	149.90 ± 0.29	158.80 ± 0.27	5.93 %
100	153.26 ± 0.65	165.73 ± 0.26	8.14 %

4.15.2 Results

Table 4.25 shows that UDP is more sensitive to scaling: increasing nodes increase latency. Table 4.24 shows UDP drift distance increasing as the node number increases (3.94 VE units at 100 nodes versus 2.45 VE units at 20 nodes). TCP drift distance remains roughly the same (5.80 VE units at 100 nodes versus 5.84 VE units at 20 nodes).

In Figure 4.15 it can be seen that the number of nodes has a small influence on the topology consistency or drift distance. The send and receive bandwidth conforms to expectation: more nodes result in more bandwidth used. The per node bandwidth increases as the node number increases: more nodes mean more interactions in the VE and therefore increase the overall per node bandwidth. The total network bandwidth (calculated by multiplying the per node bandwidth with the node count) is shown to give an indication of the total bandwidth required for running the VE.

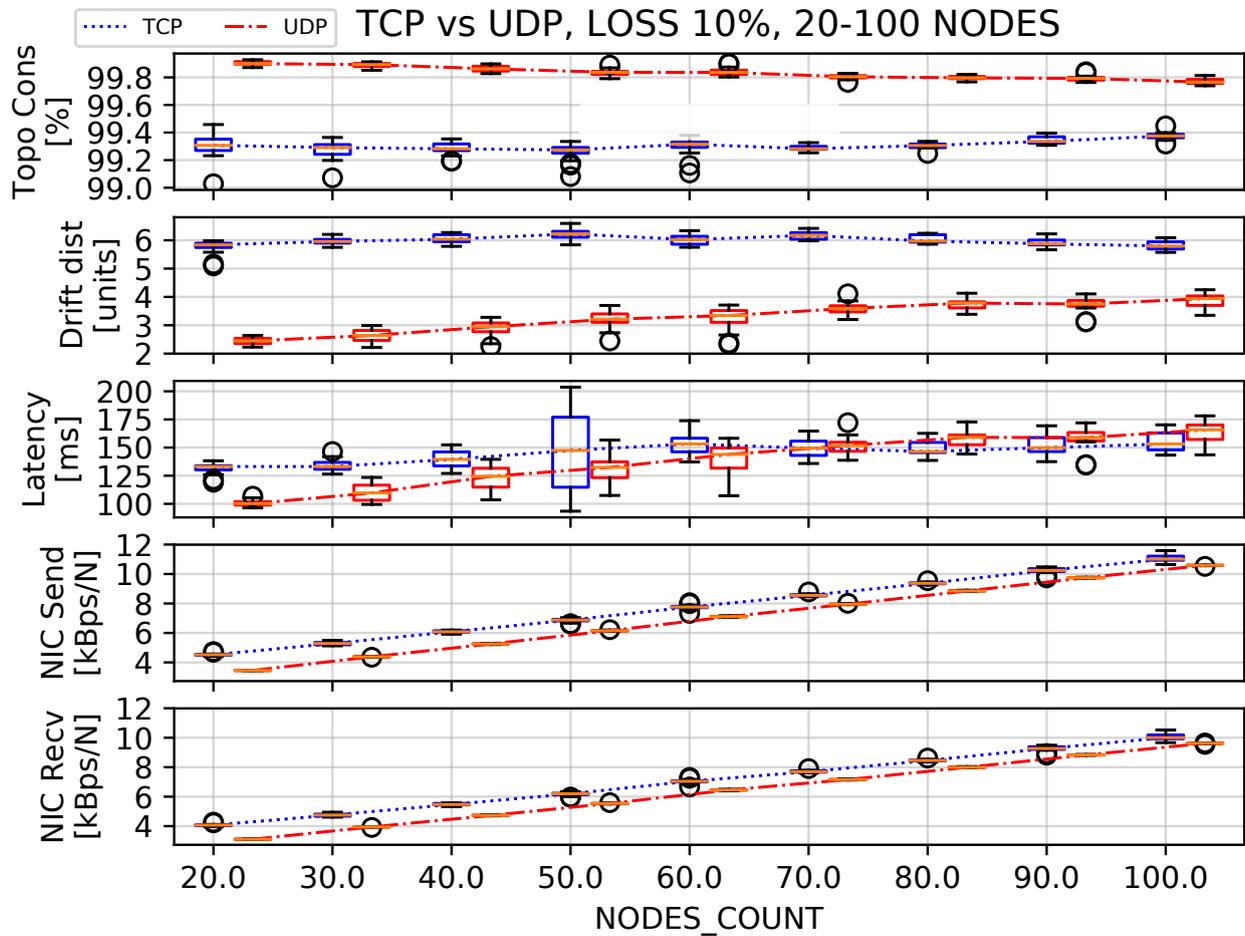


Figure 4.15: Scaling test with 20 - 100 nodes. TCP results are shown in blue and UDP results in red. All tests were run at a packet loss percentage of 10% and zero delay. Outliers are indicated with black circles.

CHAPTER 4. RESULTS OF VAST MMVE UNDER ADVERSE NETWORK
94 CONDITIONS

Table 4.26: Median of normalised NIC send bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when increasing the number of nodes.

# Nodes	TCP	total	UDP	total	\pm %
20	4.52 \pm 0.00	90.43	3.45 \pm 0.00	69.10	-23.59 %
30	5.28 \pm 0.00	158.52	4.38 \pm 0.00	131.46	-17.07 %
40	6.08 \pm 0.00	243.14	5.25 \pm 0.00	210.19	-13.55 %
50	6.87 \pm 0.00	343.41	6.15 \pm 0.00	307.63	-10.42 %
60	7.76 \pm 0.00	465.57	7.11 \pm 0.00	426.57	-8.38 %
70	8.55 \pm 0.00	598.51	7.95 \pm 0.00	556.55	-7.01 %
80	9.37 \pm 0.00	749.46	8.85 \pm 0.00	707.99	-5.53 %
90	10.24 \pm 0.01	921.87	9.75 \pm 0.00	877.35	-4.83 %
100	11.02 \pm 0.01	1102.41	10.59 \pm 0.00	1059.05	-3.93 %

Table 4.27: Median of normalised drift distances [VE units] and 90% confidence interval when increasing the number of nodes.

	TCP	UDP	\pm %
50	2.14 \pm 0.00	2.72 \pm 0.01	26.99 %
100	2.39 \pm 0.01	3.20 \pm 0.02	34.35 %

4.15.3 Discussion

Most of the results are fairly expected: the state consistency does not seem to degrade under the conditions which we were able to scale the simulation. More extensive real world testing could possibly show scaling limitations.

As was expected, more nodes in the VE require more bandwidth. The increase in required bandwidth is due to two effects: firstly, more nodes in the VE increase interaction opportunities and therefore the bandwidth. Secondly, the bandwidth per node is scaled with a larger number of nodes. These two effects compound bandwidth requirement for scaling the VE. This can cause scalability problems if large numbers of nodes are present in the VE.

4.15.4 Comparison with Earlier Papers

Although scaling tests in related papers are much more extensive (up to 1000 nodes), we can compare the trends between 50 and 100 nodes results with in the papers.

- *Scalable AOI-cast for peer-to-peer networked virtual environments* [75], Jiang, Huang and Hu present a transmission bandwidth test indicating around 8 kBps / node for 100 nodes which we can

Table 4.28: Median of normalised VAST send bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when increasing the number of nodes.

	TCP	total	UDP	total	\pm %
50	5.07 ± 0.00	253.6	5.15 ± 0.00	257.35	1.48 %
100	9.69 ± 0.00	968.82	9.63 ± 0.00	963.31	-0.57 %

compare to our 9.69 kBps / node (shown in Table 4.28). They also show a drift distance test with VoroCast (a message forwarding technique also used in VAST). Their value of around 2 VE units of drift distance for 50 nodes corresponds well to our 2.07 VE units (shown in Table 4.27) result at no packet loss.

- In *A forwarding model for voronoi-based overlay network* [44], Chen et al present a transmission size test at 50 and 100 nodes. Their values of [estimated] 4000 bps / node corresponds fairly well to our 5.07 kBps / node for 50 nodes, and 6000 bps / node and our 9.69 kBps / node for 100 nodes (shown in Table 4.28). Their AOI of 200 correspond well to our AOI of 195.

4.15.5 Summary

In this section we saw that the number of users of the VE does not significantly influence the state consistency for the scale which we could simulate. However, we saw that the property increasing most rapidly was the total network bandwidth which could eventually lead to scaling limitations.

4.16 General discussion and recommendations

In general we observed that TCP does not perform well under adverse network conditions. It is recommended to use UDP as a transport layer for networks under adverse conditions.

However, even with using UDP as transport layer, adverse network conditions such as packet loss and network delay decrease the state consistency and therefore degrades the VE experience.

4.17 Which problem can be addressed on Transport Layer?

In the above tests we observed the effect of packet loss and network delay on the state consistency of a VE. It is clear that any adverse network effects will degrade the VE experience.

In the ideal case a transport protocol is required which is completely reliable, low latency and scalable.

In our tests above we have seen that TCP, although supposedly reliable, is not well suited to adverse network conditions. UDP on the other hand deals better with adverse conditions, but is inherently unreliable and therefore cannot compensate for packet loss. It was also shown that the UDP implementation is more sensitive to scaling and experiences more processing delay as the number of nodes increase.

In a real world setup it is typically difficult (in some cases impossible) to reduce network delay. Network delay is typically a factor of physical distance, spatial channel reuse or frequency spectrum availability. Optimisations in these areas are beyond the scope of this dissertation.

Packet loss is typically a factor of physical layer hardware and MAC layer protocols. MAC layer protocols do attempt to perform layer 2 retransmissions when a packet is lost: see [66] for an analysis of layer 2 retransmission protocols.

However, as was seen in the TCP scenario above, retransmissions cause increased delay in lossy environments, degrading the VE experience. In this dissertation we will attempt to address packet loss in a novel way: using network coding for packet redundancy, adding a recovery mechanism without the need for retransmissions. In the next chapter we will discuss existing packet loss mitigation strategies and why we decided to use network coding.

4.18 Summary

In this chapter we demonstrated the effect of adverse network conditions on the state consistency. Both TCP and UDP network implementations were affected, but TCP degraded the most. We concluded that a mitigation strategy is needed and that the problem of packet loss can be addressed on the transport layer.

In the next chapter we will discuss possible packet loss mitigation strategies.

Chapter 5

Mitigation Strategies for Packet Loss

In this chapter, we will discuss various packet loss mitigation strategies. Then we will explain Network Coding and how it is another useful option for transport layer recovery. First, we will need to discuss the Open Systems Interconnect (OSI) model to understand the layers of protocols on the Internet and how recovery can be implemented on the different layers.

5.1 Open Systems Interconnect (OSI) Model

The Open Systems Interconnect (OSI) model is a conceptual model of the encapsulation processes on the Internet. The OSI model was created by the International Organisation for Standardization as a model for defining the functions of each of the layers in a network stack [25].

A simplified model referred to as the Internet Protocol suite is frequently seen in packet switched networks such as Local Area Networks and Wi-Fi. It was created originally by the Defense Advanced Research Projects Agency (DARPA) as a way of interconnecting existing networks. It is now in general networking use and facilitates the Internet (an internetworking of networks worldwide) [49].

Figure 5.1 shows the OSI layers and the Internet Protocol (TCP/IP) suite layers side by side.

In this dissertation, we will mainly discuss the Internet Protocol suite as that is the layers typically present in our network setups. In some cases, we will refer to the physical layer and data link layers separately: the physical layer is usually the hardware components (such as wireless radios and Category 7 cables) while the data link

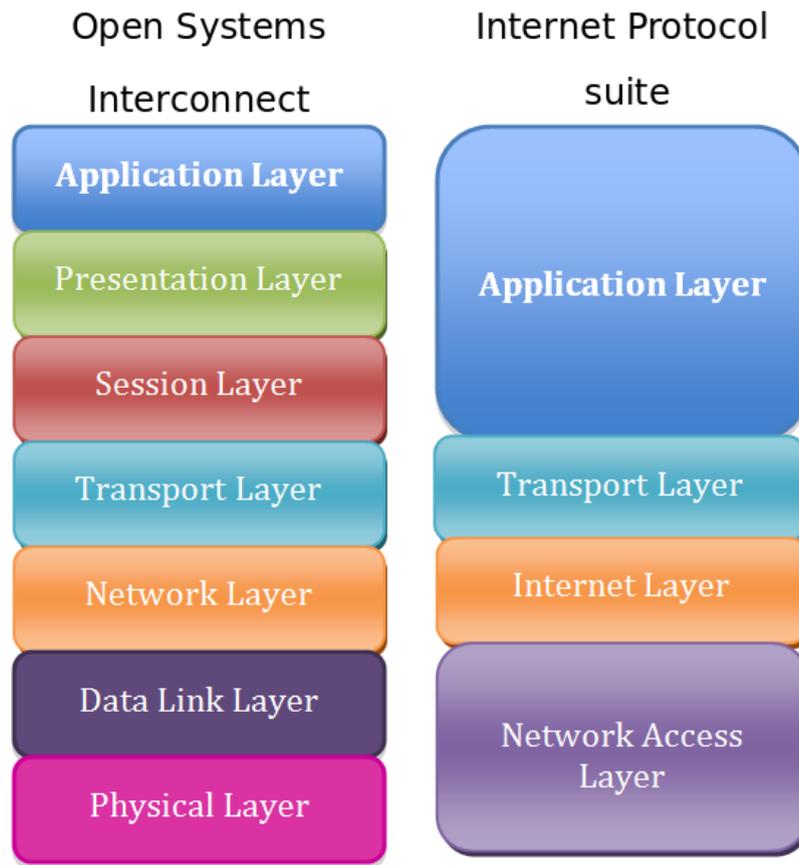


Figure 5.1: OSI model and Internet Protocol suite abstraction layers.

layer could include the Medium Access Control (MAC) and the Logical Link Control protocols (such as 802.11 Wi-Fi and 802.3 Ethernet). We will sometimes refer to the Data-link layer as the MAC layer.

5.2 Communicating Using a Network Stack

Each host in the network uses the network stack to communicate. Layers are only allowed to communicate vertically except the physical layer which is connected with wires or over-the-air to another host.

An application needing to send a string of bytes to an application on another host communicates with the layer below it. In the Internet Protocol suite this would mean that the application layer communicates to the transport layer.

The transport layer accepts the string of bytes as the payload and adds a header for specifying the endpoint. For UDP (a transport layer

5.3. THE STATUS QUO: WHAT HAPPENS IN NETWORKS CURRENTLY **99**

protocol), this would include the source and destination ports, showing at which port on the remote host the application bytes needs to be delivered. The application bytes are said to be encapsulated by the transport protocol, i.e. treated simply as a payload without knowledge of the contents.

The transport layer then communicates to the Internet layer to send the packet (containing the header and application bytes). Once again, the Internet layer treats the entire packet as the payload of its encapsulation and adds a header for specifying a destination, in this case the IP address.

Finally, the IP layer communicates to the network access layer, which once again encapsulates the packet. The physical layer hardware then produces a signal which can be sent over the wired or wireless medium. Depending on the medium and network setup, intermediate switches and routers read the packet headers and routes the packet to its intended destination.

At the receiving host, the physical layer hardware decodes the signal to a packet and removes the network access layer header, passing the IP packet to the Internet layer. This process continues until all of the headers are removed and the original application byte string is passed to the application running on the receiving host.

Now that we understand the basics of the networking stack, we can discuss adverse network conditions recovery mechanisms available on different levels of the network stack.

5.3 The status quo: What happens in networks currently?

If there is an error in transmission, one of the following scenarios can happen:

- If MAC layer CRC checks are enabled (typical for 802.11 Wi-Fi MAC and 802.3 Ethernet MAC) a bit error in the packet can be detected with high accuracy [95]. If a bit error is detected, the packet is discarded. If the MAC protocol enables retransmissions (eg. 802.11 Wi-Fi), the packet will be sent again, otherwise the packet would be lost to the upper layers.
- If MAC layer checks have been disabled, it is possible that an erroneous packet can be passed to the network, transport and application layers. IP and TCP both have checksum fields which will indicate if a packet error was detected, although these checksums are not as effective at detecting byte reorder-

ing or multiple bit errors [79]. If detected, the packet will be dropped and in the case of TCP a retransmit will happen.

- If all of the underlying protocols miss the errors, or if UDP is used, the erroneous packet will be passed to the application. Depending on the application, this erroneous packet can be used, discarded or even cause an unknown error state.

In most cases, erroneous packets are discarded at the lower layers or retransmitted. Therefore the application will either experience a gap in communications (dropped packet) or a delay (retransmission).

5.4 Recovery and Compensation Mechanisms (i.e. how can we do better?)

In the next sections, we will discuss recovery or compensation mechanisms on the physical/MAC, network and transport protocol layers based on a survey paper by Khan et al [79]. There are various protocols on the application layer that can recover or ignore network errors, but most use similar mechanisms to the other layers, so they will not be discussed here.

Khan defines three options of loss management:

1. **Perfect transmission required, retransmission allowed:** If the cost of retransmitting the packet is low, network conditions is typically favourable (loss free) or the bandwidth available is restricted, retransmissions would be the most effective strategy of handling adverse network conditions.
2. **Retransmissions expensive, error corrections allowed:** In these cases the cost of retransmission is too high, for example, multi-hop wireless networks, or low-powered Internet-of-Things devices which only wake-up for specific transmission schedules. In these cases, error correction techniques can be used to recover partially corrupted packets. It may be possible that error correction techniques cannot recover packets fully and therefore the packet loss is exposed to the application.
3. **Error tolerance:** The application layer protocols are allowed to receive corrupted packets and choose how to handle it.

5.4.1 Network Access Layer: CRC, ARQ, Error Correction Codes and Partial Packet Recovery

Many error checking codes exist, but Cyclic Redundancy Checks (CRCs) are very widespread in use as they are stronger at detecting transmission errors than usual checksums such as Fletcher's checksum and TCP's checksum [95]. CRCs are performed using polynomial division in base 2, and the remainder is added to the transmitted packet. At the receiving end the CRC remainder can be calculated which should correspond to the remainder transmitted in the packet. CRCs have been so effective in detecting transmission errors that they are used in most Internet suite protocols [79]. When a CRC indicates a failed packet transmission, a retransmit can be requested or an automatic retransmit will be performed.

Automatic Retransmission Request (ARQ) is a protocol frequently used in TCP as well as in 802.11 Wi-Fi [79]. Packets are sent with a sequence number so that the receiver can know which packets are missing. The receiver is required to Acknowledge (ACK) each received packet. If an ACK is not received within a predetermined timeout or a packet is Not Acknowledge (NACK), the packet is retransmitted. The ARQ timeout is usually related to Round Trip Time or a multiple thereof. Using CRC and ARQs, the sender can ensure that the receiver has the correct packets despite packet loss or other adverse network conditions.

As an alternative to retransmitting every corrupted packet, Error Correction Codes (ECC) attempt to recover partially received packets. Khan gives an excellent summary of the many error correction codes in use such as Reed Solomon Codes, Low-Density Parity Check codes and Raptor codes. The aforementioned codes were used in ADSL, 802.11n and 802.11ac Wi-Fi and LTE applications respectively. Their purpose is to recover a limited number of random bit errors. They can achieve this by dividing the data into code blocks and adding a parity field after each code block. Inverting the computation used to generate the parity yields the information needed to repair the corrupted packet. If the error correction code is still unable to repair all of the bit errors, a retransmission will have to be requested or the packet can be discarded. Discarding the packet will appear like a complete packet loss in the higher layers.

Finally, according to Khan the physical layer is the best place to detect and recover errors: the physical layer can possibly identify valid and invalid bits in the received packet, for example based on the adjacent channel interference measured during each bits reception. All of the bits, each marked with a confidence of correct reception, can be passed to the upper layers for partial decoding if

the upper layer protocols accept it. This is referred to as Partial Packet Recovery (PPR). If PPR is not successful or acceptable, the physical layer can request only the corrupted bits, instead of the entire packet, and thus consume less network bandwidth.

5.4.2 Network Layer: Refector

Refector [108] is a network and transport layer packet repair mechanism attempting to correct vital fields in packet headers. Although it runs on the network level, it also uses information from the transport layer, and attempts to repair headers from both layers. Heuristics are used to predict which fields could be incorrect based on existing communication flows at the host. For example, if the receiving host has two connections open with known IPs and port numbers, the incoming packet's headers are compared to the open connections and the best match is selected to repair the headers. Using the Hamming distance of each possible correction, i.e. the number of bits in the header which need to change to match communication flow's IP and port number, the best match (lowest distance) can be selected. Refector categorises the following fields of the network and transport layers as vital for successful operation: IP protocol, IP source and destination addresses and UDP source and destination ports. Only on these fields the repairs are attempted. Note that packet length is not listed, meaning that the Refector mechanisms can only be used with applications that allow errors in the packet headers and have means of determining the packet length on the application level. Refector proposes that its primary purpose would be audio and video applications such as VoIP, which can tolerate partially corrupted packet headers and payloads. Refector allows applications to specify if corrupted messages are permitted, allowing loss accepting communications to co-exist with error-free communication. If the application does not support using corrupted messages, packets are simply dropped by Refector.

5.4.3 Transport Layer: TCP, SCTP, QUIC

The Transport Control Protocol (TCP), first defined in RFC 675 [41], allows the reliable and ordered sending of byte streams. After the initial RFC document, many more improvements to TCP have been published and implemented. We will describe the functioning of TCP as is currently the standard, specified in RFC 7414 [87].

TCP achieves packet ordering by adding sequence numbers for each byte of data and using ACKs to determine if all of the bytes in the sequence have been received. If bytes in a sequence is not

5.4. RECOVERY AND COMPENSATION MECHANISMS (I.E. HOW CAN WE DO BETTER?)

103

received, it is retransmitted. If bytes are missing in a sequence, the receiving end can ACK the last sequence of received bytes to show that it still needs the bytes following it. Using cumulative ACKing, the sender now knows it should retransmit all the bytes following it. Alternatively TCP can use Selective ACKs, showing which of the bytes in the sequence was successfully received. If no ACK is received, a retransmit is performed (similar to ARQ explained above) after a Retransmission Time Out (RTO).

In order to ensure the transmission is error-free in addition to complete and ordered, a checksum is computed of all the bytes in the transmitted segment and added to the packet. The receiving side can compute the same checksum to determine if the segment is correct. According to Partridge et al. [95], the TCP checksum is less effective at detecting errors than CRC. However, TCP checksums are still considered valuable in detecting in-host software errors in network adapter and drivers. Combining link layer CRC and TCP checksums provide sufficient checking for end-to-end data transmission [79]. TCP was one of the first transport layers used on DARPA NET and is still used in many applications today [52].

A recent alternative to TCP is the Stream Control Transmission Protocol (SCTP) [18]. SCTP is a *message orientated*, reliable transport layer protocol. Similar to TCP, SCTP provides reliable transmission by assigning sequence numbers and requiring ACKs for each sequence number. However, instead of using a sequence number for each byte transferred, data is divided up into chunks for transmission and each chunk is assigned a sequence number. If a gap in sequence numbers is detected, an SACK is sent to notify the sender of missing chunks. The sender can then retransmit the required chunks. Similar to TCP, retransmissions can also be started because the RTO expired.

SCTP bundles chunks in SCTP packets, depending on the Maximum Transmission Unit (MTU) available on the lower layers.

In SCTP, messages define the transfer actions instead of byte streams. That is, instead of an infinite stream paradigm (like TCP), SCTP uses a message paradigm (like UDP). The TCP paradigm is useful for example when transferring a very large file, while the SCTP paradigm is useful when transferring many individual records, as the records are already demarked into individual messages.

Another feature of SCTP is multi-homing capability: using multiple paths to reach the destination, possibly using two interfaces (such as mobile and Wi-Fi). This can be used for reliability, i.e. both paths carry the same information and ordering is carried out on the receiving side, discarding duplicates. Data chunk sizes are determined by the smallest MTU of both paths.

SCTP provides a valuable alternative (although not replacement) to TCP for the following reasons:

- Chunk based retransmissions are more effective and
- multi-homing support can provide extra reliability.

However, according to [24] the reasons why SCTP is not more widely used is as follows:

- TCP is already well established for many years.
- SCTP requires changing of IP stacks and applications.
- SCTP is still relatively unknown.
- Many firewalls on business and private home networks are not enabled to allow SCTP traffic.

QUIC [73] is a UDP based transport protocol suggested as a possible alternative for carrying the Hyper Text Transmission Protocol (HTTP). It was created and is used by many Google services [31]. It leverages the lower latency characteristics of UDP, but adds flow control, multi-stream support, ordering and encryption functionality. Multiple streams are particularly useful when carrying web page traffic as different elements can be sent concurrently and packet loss in one element will not delay transmission of the other elements. Error detection and retransmission can be enabled for QUIC, functioning very similar to TCP's fast retransmit.

QUIC can dispatch packets in byte stream format (such as TCP) or individual messages (such as UDP). The Internet Engineering Task Force (IETF) is currently working on a standard, but it is still in draft format.

5.4.4 Summary of Mitigation Strategies

In this section, we saw that error correction or compensation can happen on most layers. In most cases there is a trade-off between retransmission delay or extra bandwidth for reliable communications. PPR and Reflector are exceptions to this trade-off as they efficiently use side information in order to perform corrections.

Table 5.1: Summary of Mitigation Strategies

	Purpose	Disadvantages
CRC	High reliability error detection	
ARQ	Automatic retransmit	Retransmission delay
ECC	Bit error correction, recovers packet without transmission	Parity bits require more bandwidth
PPR	Interference estimation for partial packet use, partial retransmit	Requires adjacent channel monitoring
Reflector	Partial IP and TCP header corrections	
TCP	Reliability, ordering, flow control and retransmission	Extra overhead, retransmission delay
SCTP	Reliability, ordering, block retransmission	Extra overhead, retransmission delay
QUIC	Flow control, ordering, multi-stream support and retransmission	Draft IETF standard, could be in widespread use in the future

5.5 Requirements And Considerations of Packet Loss Mitigation Strategy

Requirements:

1. Network delay needs to be as little as possible as this would minimise drift distance. Therefore retransmission strategies would not be allowed.
2. Processing delay needs to be limited so as to not interfere with normal VAST operations.
3. Like VAST, the mitigation strategy needs to be distributed, i.e. no centralised coordination should be necessary.

Considerations:

1. Do not want to change Physical / MAC layer. These layers are typically implemented into router hardware and would therefore be difficult to change as well as reduce the interoperability to other devices.
2. As far as possible, we would like to keep the original VAST implementation in order to leverage the VAST scaling and interest

management capabilities as well as allowing comparisons to previous work.

From the reasons above and TCP results in Chapter 4, it is clear that we cannot use retransmission strategies. Ideally we would like to perform error correction, but we do not have access to the physical layer and therefore would experience packet loss and not receive corrupted packets. This also corresponds to how Mininet presents packet loss: packets are dropped as a whole and not partially or corrupted. Therefore we do not consider error correction codes, but rather focus on erasure codes, that is, ways of recovering a full packet lost.

5.6 Naive Erasure Recovery

The simplest recovery from packet loss is resends. In TCP resends are used when the packet is not acknowledged, limiting the extra recovery bandwidth at the cost of increased delay. However, the acknowledgements could be completely removed if *all* packets are simply sent twice or even multiple times. This would reduce the effect of packet loss because the probability that multiple repeated packets are lost is lower than single packet loss. In fact, this is one of the improvements made in [98] where unacknowledged packets are resent before a retransmission was requested or scheduled.

The downside of this strategy is the effective doubling or more of sending bandwidth which could cause congestion. In [98] they argue that the redundant sending of packets is worthwhile as the bandwidth requirements of VE applications are typically much lower than network capabilities (so called thin-streams) and therefore the bandwidth is available.

In the following sections we discuss more efficient erasure codes. Although we eventually select network coding as an erasure code, it could be useful future work to compare network coding with this naive recovery strategy.

5.7 Erasure Codes

In this section, we will give a brief discussion of erasure codes. This is not meant to be an exhaustive list, but rather a few example codes that are in practice today. Although Low-Density Parity-Check and Reed-Solomon codes are generally classified as error-correcting codes, they can efficiently be used as erasure codes as well.

5.7.1 Low-Density Parity-Check Code

Low-Density Parity-Check (LDPC) [61] codes are linear error correction codes created by Robert Gallager in 1962. The codes are generated by appending parity bits, i.e. ones or zeros in order to make the sum of data bits **even**, to input bits, similar to a Hamming code [91]. The parity check matrix represents the linear equations used for calculating the parity combinations. The parity check matrix is used to generate the parity bits from the data bits, forming a codeword. Unlike Hamming codes, the parity check matrix can be pseudo-random and should be *sparse*, that is, only a small number of bits should be ones.

Encoding is done by multiplying k input bits with the $r \times n$ parity check matrix in a finite field, with n codeword length and $r = n - k$ parity bits. Decoding can be done by belief propagation algorithms on a graphical representation of the parity check equations. By using the error probabilities received from the channel as well as the probabilities of bit error based on the parity equations, an iterative process of changing the bits most likely to be incorrect, a codeword satisfying all of the parity check equations can be obtained. For more information about decoding techniques, please refer to [43].

LDPC codes achieve very near the Shannon limit for information transfer rate for a given channel noise level [111]. LDPC codes are used in Digital Video Broadcasting and are included as optional in the 802.11n and 802.11ac Wi-Fi standards.

5.7.2 Reed-Solomon Code

Reed-Solomon codes are a block linear codes which were created by Irving Reed and Gustave Solomon in 1960. It is used in CDs, DVDs, RAID-6 storage, satellite communication and ADSL. A Reed-Solomon code is constructed from k data symbols (typically bytes as symbols), adding parity symbols to form a n symbol codeword. The number of parity symbols added determines the number of symbols that can be corrected. The number of parity symbols is:

$$p = n - k = 2t \quad (5.7.1)$$

where p is the number of parity symbols. It is guaranteed that t symbol errors can be corrected with a Reed-Solomon code. Encoding is achieved by dividing (in a finite field) the data symbols by a irreducible generator polynomial, and storing the remainder. The rank of the generator polynomial ensures that p parity symbols are present in the remainder. The parity symbols are appended to the original message to form the codeword.

The decoder calculates the syndromes of the code, that is the value of the received polynomial evaluated at the roots of the generator polynomial. If no transmission error occurs, the syndromes will all be zero. However, if a transmission error occurred, non-zero values in the syndromes will show this. Once it is clear that there were symbol errors, a locator polynomial and evaluator polynomials can be calculated based on the syndromes. The locator polynomial will tell us *where* the symbol errors occurred and the evaluator polynomial will tell us *how* errors occurred. Together with the syndromes, this will allow us to calculate the magnitude polynomial which needs to be subtracted from the received polynomial to yield the original message. See [122] for a more detailed description.

5.7.3 Luby Transform Code

Luby Transform (LT) codes are fountain codes initially presented by Michael Luby in 1998. Fountain codes are rateless, meaning that unlike Reed-Solomon codes, there is no fixed ratio between the original blocks and the redundant blocks. Put differently, rateless codes can produce an infinite number of coded blocks. Only a number of blocks slightly larger than the original message length need to be received in order to decode the original message.

The encoding and decoding use the XOR operation to code the blocks together.

In LT codes, the message is divided into K blocks of equal length. To generate the encoded blocks, a degree d , and then d indexes are chosen pseudo randomly. The blocks at the chosen indexes are XOR'ed together. The blocks are then sent to the receiver. Encoded blocks are generated for as long as the receiver does not acknowledge the complete reception of the message.

On the receiving side, a buffer is kept with encoded blocks. When a new block is received, it is XOR'ed with every other block in the buffer *which it shares blocks with*. As the XOR operation acts as a subtraction in a binary finite field, this means that the remaining number of encoded blocks in the result from the XOR will be reduced. As soon as there is a encoded block containing only one original block, it is used to reduce the number of blocks in the remaining encoded blocks. The process continues until all of the blocks of the message is decoded.

LT codes reduce the computational complexity of decoding blocks by using XOR operations as well as using sparse encoding degrees, i.e. the generated degree d is restricted to small values.

LT codes encoding and decoding costs scale with $K \log_e K$ and can be decoded with roughly $K + 5\%$ blocks. See [89] for an excellent

description of LT codes and Raptor Codes.

5.7.4 Raptor Code

Raptor Codes use LT codes in conjunction with LDPC codes in order to further reduce encoding and decoding to near linear complexity. This is achieved by using optimised LDPC codes with an LT code with a very low degree (typically $d = 3$). LDPC can correct erasures and can encode and decode in linear time and LT codes can be decoded very quickly if the degree is low.

The encoding is performed by using LDPC on the message (of length K) in order to produce $\tilde{K} = K/(1 - \tilde{f})$ symbols, where \tilde{f} is the expected erasure rate of the channel. Then LT codes are used to generate redundant packets from the LDPC symbols. In a LT code with low degree, not all of the blocks are covered, which will lead to missing blocks on the receiver side. However, because the message was pre-coded with LDPC, these missing blocks can be recovered.

A Raptor code implementation, RaptorQ, is specified in RFC 6330 [16] and can provide 99% reliability with K symbols received and 99.9% reliability with $K + 1$ symbols received.

Raptor codes are already used in 3G mobile networking.

5.7.5 Summary

In this section, we discussed erasure codes already in use today. Most erasure codes attempt to limit decoding and encoding time while still providing a high reliability. Typically a trade-off between performance and reliability needs to be made. It is clear from the Raptor code section that combining codes can yield even better performance without significantly reducing reliability. It is therefore useful to investigate a diverse range of codes, potentially uncovering combinations that could provide useful properties.

In the next section, we will discuss Network Coding, which can be used as an erasure code and functions similar to LT codes. However, the biggest difference between Network Coding and all of the above mentioned codes is that network codes can be applied within the network while the above codes are end-to-end codes.

5.8 Network Coding

Network coding (NC) was introduced by Ashlweide et al. [30] to increase network bandwidth to the theoretical maximum given by the Max-flow Min-cut theorem [8], i.e. improving throughput. In this

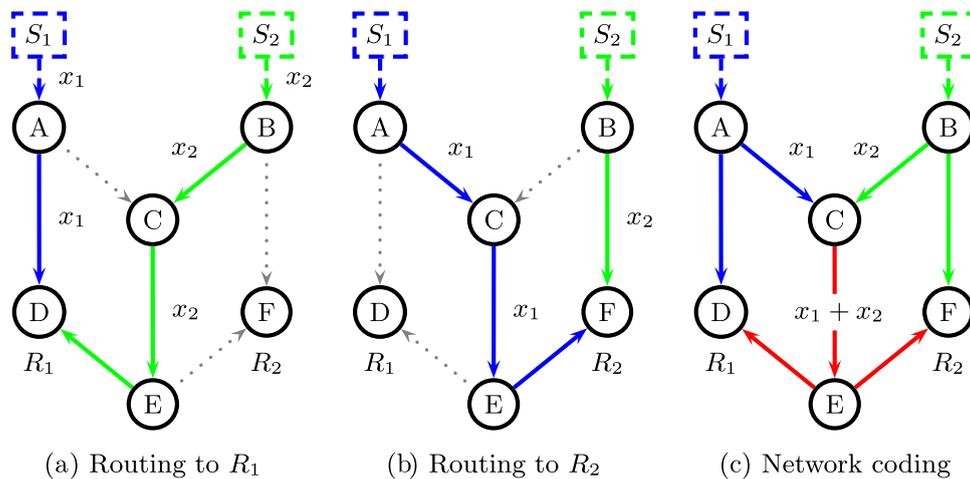


Figure 5.2: Network Coding in the Butterfly Network, *Source: [59]*

section, we will discuss how network coding can improve throughput and how network coding can be used to increase reliability by acting as an erasure code. Then we will present an algebraic framework which can be used to describe a network coding scenario, Random Linear Network coding, inter-flow and intra-flow network coding. We will also discuss the potential bandwidth, latency and computational costs associated with using network coding as an erasure code.

Finally we will discuss network coding applications in related work.

5.8.1 Throughput Improvement

We will use Figure 5.2 to explain the Max-flow Min-cut theorem. In the figure we can see two senders S_1 and S_2 , located on nodes A and B respectively. R_1 and R_2 represent the two receivers, located on nodes D and F. There are two outgoing links from node A and from node B. Using these links, two *independent* paths can be constructed between any sender and receiver. Put differently, it would take the removal of at least two links to completely disconnect a sender from the receivers. The minimum number of cuts needed to disconnect a sender from a receiver is their min-cut criterion. If each link can carry one bit in each timestep (unit capacity links) and assuming no delay links, a maximum of two bits can be delivered to R_1 in one timestep by using paths [A-D] and [A-C-E-D]. Therefore the maximum flow of information is also two bits between S_1 and R_1 if *no one else* is using the network. This corresponds to the Max-flow Min-cut theorem: The theoretical maximum flow of information be-

tween a source and a sink is equal to the number of links that need to be cut to disconnect the source from the sink.

However, in general networks links are frequently shared. Figure 5.2a) and b) show the example routing paths from S_1 and S_2 to the receivers. The shown paths are not the only paths available, but rather the most efficient paths. As we mentioned above, two independent paths can be constructed from each sender to each receiver, though not all of them are shown here. From the blue lines in the figure, we can see that there are only one set of independent paths from S_1 to both R_1 and R_2 . Therefore the maximum information flow per time slot between from S_1 is 1 bit to *both* receivers. Similarly for S_2 . Other configurations of the paths can be constructed, but the result will be the same.

It can be seen that in this network there will be contention on the link [C-E] if both senders would like to communicate *simultaneously*. Therefore under normal circumstances, the senders will have to alternate using the link. The flow rate between $S_1 \rightarrow R_2$ and $S_2 \rightarrow R_1$ is now reduced to 0.5 bits per timestep per flow.

This is where network coding can help restore the information flow to the theoretical maximum. By encoding the two packets x_1 and x_2 together, producing one symbol $(x_1 + x_2)$, the link [C-E] can send the combined symbol. The combined symbol is forwarded via [E-D] and [E-F]. R_1 now has symbols x_1 and $(x_1 + x_2)$ and R_2 has x_2 and $(x_1 + x_2)$. If R_1 is able to extract x_2 from $(x_1 + x_2)$ by subtracting x_1 , it will have received both symbols; similarly for R_2 . The rate between the senders and *both* receivers have now been restored to 1 bits/timestep.

Similar explanations can be made for wireless network coding, shown in Figure 5.3. In wireless networks, the communications medium is typically shared, i.e. only one node can communicate at a time. For a message to be communicated from node A to C, two time slots are needed: A to B and B to C. This can be described as a maximum information flow of 0.5 bits per time slot between A and C. If both A and C want to exchange messages, four time slots are needed as can be seen on the left hand side of the figure. Despite the shared communication channel, the information flow is still 2 bits/-four timesteps = 0.5 bits/timestep. However, with network coding we can improve upon that by using the broadcast nature of wireless. If the transmission is timed correctly, the symbol $(x_1 + x_2)$ can be sent in the final timestep shown on the right side. This results in 2 bits/3 timesteps, or 0.66 bits/timestep or an improvement factor, also called a *coding gain* of 1.33.

From the two examples above, it is clear why it is called *network coding*: encoding of symbols are allowed to happen *within* the net-

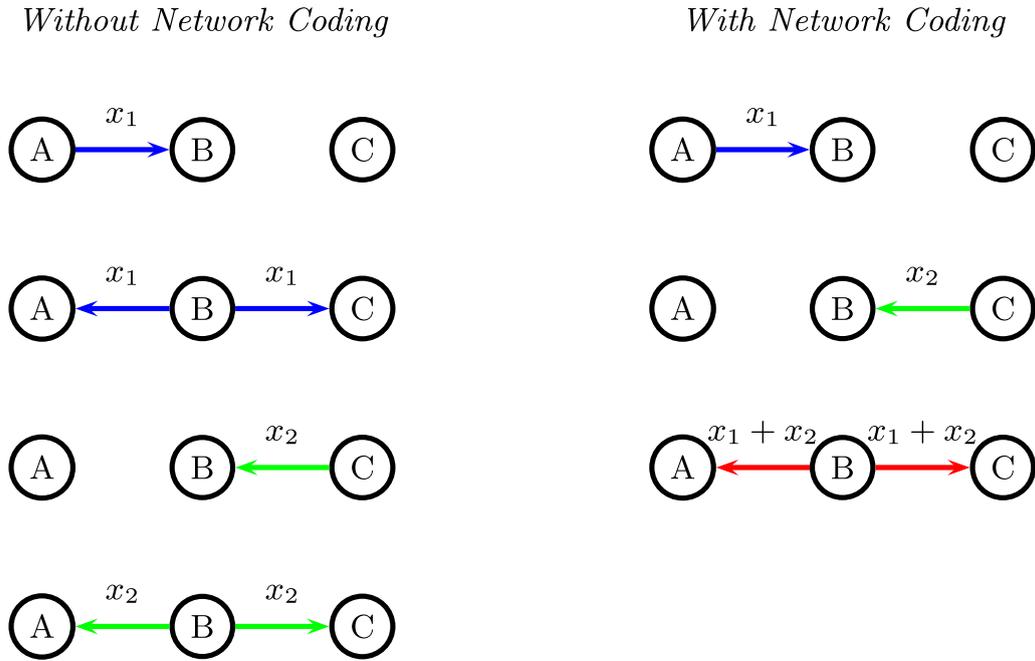


Figure 5.3: Wireless Network Coding, Source: [59]

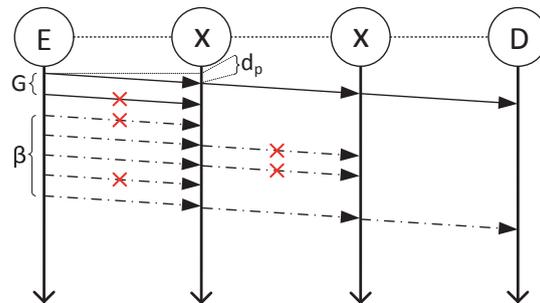


Figure 5.4: End-to-End Coding, Source: [115]

work.

5.8.2 Reliability Improvement

Since that initial paper, network coding has also been proposed to increase reliability in lossy networks, amongst others in a paper by Szabo et al. [115]. We explain the coding processes mentioned in the paper shortly.

Consider the network setup shown in Figures 5.4,5.5 and 5.6

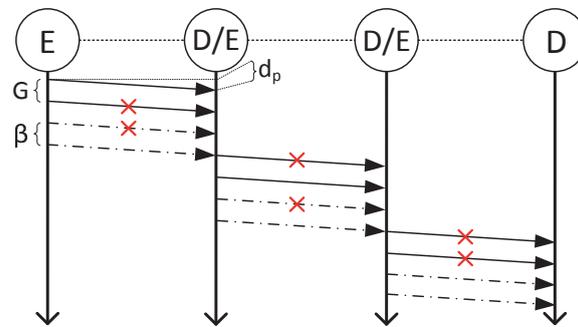


Figure 5.5: Hop-by-Hop Coding, Source: [115]

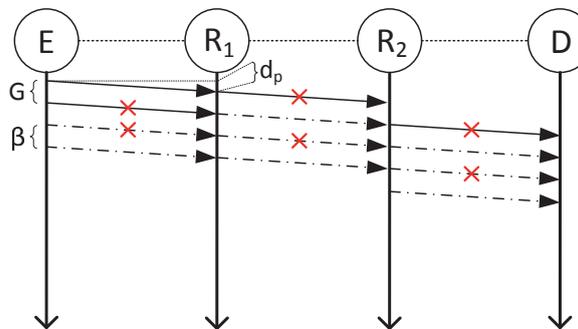


Figure 5.6: Random Linear Network Coding, Source: [115]

with transmission happening across the four nodes at the top. In this network there is a probability η that the packet transmission between nodes can fail. Below the nodes are a sequence diagram of the transmissions. A line with a red cross indicates a failed transmission. The downward slope indicated the time taken to transmit the message between nodes. G indicate the two blocks of the message that need to be sent. β indicates the redundant blocks that were required to be sent for successful transmission.

Three approaches were evaluated: End-to-End (E2E) encoding, Hop-by-hop (HbH) encoding and Random Linear Network Coding (RLNC), a type of network coding, discussed in more detail below. The nodes are labeled as follows: **E** shows an encoding node, **X** shows a forwarding node, **D** shows a decoding node and **R** shows a recoding node. Recoding is an operation only available in network coding.

In the E2E and HbH approach a erasure coding scheme such as Raptor codes can be used to generate redundant blocks. If a fountain code is used, more codewords can be generated as needed. In RLNC, Network Coding can be used to generate redundant blocks, such as the $(x_1 + x_2)$ in the previous subsection, in addition to the the original x_1 and x_2 symbols already transmitted. In this way RLNC can also be seen as a fountain code.

In E2E (Figure 5.4), the original blocks are encoded with an erasure code to generate all the codewords that will be transmitted. It can be seen that the first transmission succeeds, and is also successfully forwarded to the endpoint. However, in order to receive the message successfully, the other block or one of the redundant blocks need to be received successfully. A transmission on any of the links can fail, therefore the message and redundant blocks require multiple transmissions to arrive successfully at the endpoint. In this way the packet loss probability is compounded. Only once two blocks are successfully received can the blocks be decoded and the original message recovered.

In HbH (Figure 5.5), the original blocks are also encoded similar to E2E. However, the blocks are decoded at each intermediate node and re-encoded. This requires that each of the intermediate nodes receives all of the blocks successfully before it can be decoded, guaranteeing that the message is correct at each node. In this way the error probability only applies to one link, instead of it being compounded as with E2E. However, the intermediate receptions, decoding and encoding produces extra delay.

In RLNC (Figure 5.6), the message can be recoded at each intermediate node and still be decodable at the endpoint. Therefore the independent nodes can send blocks as soon as it receives any. As blocks are received, different combinations of the received blocks are coded together and transmitted. This is called *recoding* as the blocks are not decoded, but already encoded or even uncoded blocks are encoded together. Like other fountain codes, any two of these blocks allow the decoding of the original message. From Figure 5.6 it is clear that RLNC can reduce the total transmission time.

Once again note that the encoding happens within the network. This also reduces the network delay to a minimum as the *network coded* packet is generated closer to the end destination than usual end-to-end coding.

5.8.3 Algebraic Framework for Network Coding

In order to understand how network coded packets are encoded and decoded, we present the algebraic framework for network coding

(based on [59]).

Let us consider a network where unit capacity links are used. That means each link in the network holds (transmits) exactly one symbol per timestep. The algebraic framework describes the network as a linear system mapping inputs from sources, via the network links, to the receivers. In this way the network topology is condensed to a network transfer matrix and output transfer functions. The symbols on each link is considered the state variables of the network.

Using the network transfer matrix, the state update equation (that is, how the symbols move through the network) is given as:

$$\hat{s}_{k+1} = \mathbf{A}\hat{s}_k + \mathbf{B}\hat{u}_k \quad (5.8.1)$$

where \hat{s}_k is the symbols in the network at time k , \hat{s}_{k+1} the symbols at the next timestep, \mathbf{A} the network transfer matrix, \hat{u}_k the input symbols and \mathbf{B} the input mapping.

The output transfer function at receiver R_j is given as:

$$\hat{y}_{k,j} = \mathbf{C}_j\hat{s}_k + \mathbf{D}_j\hat{u}_k \quad (5.8.2)$$

where $\hat{y}_{k,j}$ is the output symbols at time k and receiver R_j , \mathbf{C}_j the output mapping, and \mathbf{D}_j the direct mapping of input symbols to the output.

If the network contains m links, the dimensions of \hat{s}_k is $m \times 1$. If h is the min-cut of each receiver, \hat{u}_k is $h \times 1$. $\hat{y}_{k,j}$ is a $h \times 1$ output vector.

The \mathbf{A} matrix represents how network links are connected, i.e. the algebraic representation of the network topology.

The \mathbf{B} matrix maps the input symbols from the sources to the links in the network, indicating how symbols enter the network. Similarly, the \mathbf{C}_j matrix maps the network symbols to the receiver R_j , showing how symbols are obtained from the network.

The \mathbf{D}_j matrix represents connection of inputs to outputs at R_j , representing a receiver connected directly to a source.

We can combine the equations 5.8.1 and 5.8.2 to obtain the transfer function for each receiver R_j :

$$\mathbf{G}_j(\Delta) = \mathbf{D}_j + \mathbf{C}_j(\Delta^{-1}\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \quad (5.8.3)$$

where Δ is the intermediate delay operator. Using unit delay links, i.e. a symbol takes one timestep to transverse a link, $\Delta = 1$. Therefore the equation becomes:

$$\mathbf{G}_j = \mathbf{D}_j + \mathbf{C}_j(\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \quad (5.8.4)$$

G_j describes the coding coefficients that need to be applied at each coding point. Coding points are defined as links where contention would happen in store-and-forward networks. At the coding points, symbols need to be combined in order to resolve these contentions and restore the full transmission bandwidth given by the Max-flow min-cut theorem.

Suppose we have h source symbols ($\sigma_1 \dots \sigma_h = \hat{\sigma}$ from source S_1, \dots, S_h) to transmit. At each coding point (link e) we will have to determine a coding vector \hat{c}_e , consisting of $c_{e,1} \dots c_{e,h} = \hat{c}_e$ coefficients.

\hat{c}_e will be multiplied by the source symbol vector σ to produce a coded symbol ρ_e .

$$\rho_e = \hat{c}_e \hat{\sigma} = [c_{e,1} \ c_{e,2} \ \dots \ c_{e,h}] \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_h \end{bmatrix} \quad (5.8.5)$$

Note that not every intermediate node will receive all source symbols, and therefore $c_{e,i}$ for those symbols will be 0.

These coding vectors are the rows of the matrix G_j . The coded received symbols at receiver R_j are $[\rho_1^j \ \rho_2^j \ \dots \ \rho_h^j] = \hat{\rho}_j$, where $\rho_1 \dots \rho_h$ represent the individual coded symbols from links $e_1 \dots e_h$. Note that a receiver would not be linked to every node in the network, and therefore ρ_i would be zero in such cases.

The receiver R_j must solve the following linear equations to retrieve the source symbols:

$$\begin{bmatrix} \rho_1^j \\ \rho_2^j \\ \vdots \\ \rho_h^j \end{bmatrix} = G_j \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_h \end{bmatrix}$$

Let us refer back to the butterfly network in Figure 5.2, as an example of G_j . In the butterfly network we have two receivers (R_1, R_2) and one coding point, link CE.

In order to describe the encoding vectors in terms of the algebraic framework, we will use α_i as placeholder coefficients in the coding equations:

$$\text{For } R_1: G_1 = \begin{bmatrix} 1 & 0 \\ \alpha_1 & \alpha_2 \end{bmatrix}$$

$$\text{For } R_2: G_2 = \begin{bmatrix} 0 & 1 \\ \alpha_1 & \alpha_2 \end{bmatrix}$$

where α_i are the required coefficients to assure decodability at each receiver. For the butterfly network, the values of $\alpha_1 = \alpha_2 = 1$ is a valid solution for the coding vector.

The algebraic framework allows us to predict if a network coding solution is valid, that is, decodable at every receiver as well as improving the total throughput to the theoretical max-flow min-cut limit. According to Network Coding's Main Theorem:

In linear network coding, there exist values over a large enough finite field \mathbb{F}_q for the components α_i of the coding vectors, such that all matrices G_j , $1 \leq j \leq N_{recv}$, defining the information that the receivers observe, are full rank [59].

N_{recv} is the number of receivers.

A finite field is a subset of the real numbers \mathbb{R} with a finite number of elements. Arithmetic operations are redefined for finite fields such that the result of addition, multiplication, subtraction and division of field elements also lie within the field. A finite field is specified as \mathbb{F}_q , where $q = p^k$. p^k is the number of elements in the finite field and p is the field characteristic.

A simple way of constructing a finite field is by performing modulo arithmetic, that is, applying the modulo operator after each calculation. A common field characteristic is 2; it is particularly useful in computer applications as addition becomes the XOR operator and multiplication the AND operator.

A large enough finite field, as required by the Main Theorem, is a finite field with enough elements to use as coefficients in the code vectors such that the packets are uniquely identifiable and decodable at every receiver.

In order to determine if a set of coding vectors are a valid solution for a network coding scenario, the matrices G_j can be checked if they are full rank. This check can also be achieved by checking the product of all G_j -matrices' determinant:

$$\prod_{j=1}^N \det G_j \neq 0 \quad (5.8.6)$$

Thus, the algebraic framework for network coding allows us to determine the coding vectors from a network topology or determine if a network coding strategy is valid for a given network.

5.8.3.1 Random Linear Network Coding

In the initial paper by Ho et al. [67] on Random Linear Network Coding (RLNC) they used the algebraic framework to describe their network coding strategy. However, instead of performing the needed

calculations in order to determine if a given coding matrix is valid, Ho proposed that the coding matrix be generated from an independent and identically distributed (i.i.d.) random process in the finite field.

Then, using the validation test for G_j , Ho derives an expression for the likelihood that a RLNC encoded symbol can be decoded as:

$$\text{Decodable} = \left(1 - \frac{N_{recv}}{q}\right)^\eta \quad (5.8.7)$$

where N_{recv} is the number of receivers, q the finite field size, and η the number of coding points. From this equation we can see that the more receivers that need simultaneous packet deliver, the less chance that we can get a valid RLNC that will be decodable everywhere. In contrast, increasing the finite field size q , we can increase the probability of decodability (similar to the Main Theorem of Network Coding). Finally, the more nodes that perform network coding, the more likely it will be that a packet will be undecodable at a receiver. All of the above statements can be explained from the linear equations mentioned above. If the coefficients of the linear equations are generated from a random process, there is a chance that coefficients will be chosen such that we have linear dependent equations. More coding nodes increase the number of linear combination symbols in the network. Receivers in the network receive different combinations, therefore having to satisfy *all* possible combinations reduces the likelihood that it will be decodable at *every* receiver. If a packet is undecodable, the more original symbols or more *differently coded* linear combinations should be sent.

The random coefficients are generated at each node therefore making the coding process completely distributed according to our requirement criteria number three.

5.8.4 Kodo Library

In order to simplify RLNC encoding and make it accessible, the Steinwurf organisation created the open source Kodo RLNC library [96]. Kodo provides an API with all the functionality for performing encoding and decoding in a finite field.

Kodo has bindings for C, C++, Python, JavaScript, and GoLang. In our implementation we will use the C++ library as the VAST library is also written in C++. The Kodo library was also used in the Szabo paper [115], showing that Kodo RLNC can be used to improve reliability and mitigate packet loss.

5.8.5 Inter-flow NC and Intra-flow NC

Network coding can either be used to improve throughput to the theoretical Max-flow min-cut rate or increase reliability and reduce the transmission time. Seferoglu et al. [109] labeled these two use cases of network coding as inter-flow NC and intra-flow NC respectively:

Inter-flow NC improves network throughput by encoding different flows (a packet route defined by a source and destination) together in such a way that both destinations can decode the NC packet. The single encoded packet is then sent over the constrained links. This approach effectively sends less packets overall.

Intra-flow NC improves network reliability by generating redundant NC packets from the *same* flow. In contrast to inter-flow NC this increases the number of packets that are sent over the network.

5.8.6 Summary

In our application, we will be using RLNC as an erasure code in order to improve reliability. Because our application is time-sensitive, we will not be using end-to-end erasure codes, but rather RLNC in order to reduce transmission time, as was shown in the work by Szabo et al [115].

5.9 Cost of Using Intra-flow Network Coding

There are three costs that are associated with using intra-flow network coding: extra bandwidth, extra decoding delay and computational complexity.

5.9.1 Bandwidth Requirements

As with any erasure code, network coding also uses more bandwidth in order to achieve reliability. In network coding this extra bandwidth manifests itself as extra symbols that can be used to recover from lost symbols. For RLNC, the coefficients of the linear equation are also attached to the encoded packet, resulting in a small added overhead. However, the major contribution of the extra bandwidth required is to send the redundant symbols. It is possible to decode the original symbols from any set of two symbols, so the minimum bandwidth for decoding the symbols are not increased. However, in order to replace the symbols lost in transmission, the extra packets

are required. As we do not know which packets will be lost in transmission, we add redundancy for all packets. The cost of network coding is thus the wasted bandwidth of redundant packets never used in the recovery process.

Note that in section 5.6 we discussed the naive erasure recovery and the increased bandwidth associated with it. Compared with the naive approach, network coding would require less bandwidth because combined packets can be used by multiple hosts, and therefore each packet will not have to be sent multiple times, but rather only the combined packets. We describe this as redundancy efficiency.

5.9.2 Decoding Delay

In intra-flow network coding, packets from the same information flow is coded together for redundancy. Note that this means that packets from earlier and later in the flow are encoded together randomly. Consider the following scenario:

Packets 1 to 10 need to be sent. After the original packets are sent, the redundant packets are dispatched. Redundant packets are coded together as follows (chosen randomly): $3 \oplus 7$, $1 \oplus 5$, $2 \oplus 10$ and $5 \oplus 8$. Now suppose that packet 2 was lost in transmission. Packet 2 can be recovered from $2 \oplus 10$ and 10, but only when 10 and $2 \oplus 10$ are received successfully. In a single block transmission like a large file, this is not a problem. However, if there is a time sensitivity related to the sequence of packets, the reception of packet 2 will be delayed until after packet 10 is received. In audio or video streaming applications this could be detrimental to the experience.

In order to reduce the longest delay times, the concept of generations are introduced. Instead of encoding packets from anywhere in the transmission queue, the queue is divided into sections called generations and encoded is only allowed within a generation. Consider the scenario above with generations applied to transmission: Packets 1-3 is generation 1, 4 - 6 generation 2 and 7-10 generation 3. Original packets from a generation are sent and redundant packets for that generation is then sent immediately afterwards. Then the next generation's packets are sent. The encoded symbols can only contain symbols from the same generation, for example $2 \oplus 3$, $4 \oplus 6$, $7 \oplus 8$ and $8 \oplus 10$. In this case, if packet 2 is lost, it only has to wait for packet 3 and the encoded packet, to be recovered.

5.9.3 Computational Complexity

Encoded symbols are defined by a set of linear equations in terms of the original symbols and therefore recovering a symbol is achieved by solving the equations for the required symbol. The number of operations needed to solve a set of linear equations in general are related to the $O(S^3)$ where S is the number of symbols in the set of linear equations [59]. This could be very limiting for low power devices that do not have the required hardware to efficiently decode symbols. There are two strategies for reducing the computational complexity: using smaller generation sizes and using a sparse encoding matrix. Both will be explained below.

In addition to reducing decoding delay, another advantage of generations is it limits the number of symbols S in the linear equations to those that are contained in the generation, thereby reducing the computational complexity.

Sparse encoding matrices (i.e. G_j containing the encoding coefficients of the linear equations) are matrices that contain many zeros and very few coefficients. That means that fewer symbols are coded together, limiting the number of symbols S in each equation.

The trade-offs between bandwidth, decoding delay and computational complexity are considered here.

Suppose that redundancy is required for each generation to improve reliability. Ideally, generation sizes would be as small as possible to reduce decoding delay and computational complexity. However, that would mean that every two or three symbols would need to be coded together. This would result in a large number of encoded symbols to be generated and transmitted, leading to higher bandwidth use.

In contrast, let us consider larger generations: the delay and complexity would be higher than smaller generations, but the encoded symbols could cover more symbols and still provide redundancy, thereby reducing the redundant packet bandwidth.

These trade-offs need to be considered on an application-by-application basis. Some applications, like video streaming would require low latency, and would therefore utilise small generation sizes. Other applications like file transfers are not as latency sensitive and can use larger generations. The packet loss rate should also be taken into consideration, as larger generations would need to generate more redundant symbols to compensate for high packet loss, which could bring it on par with smaller generation sizes bandwidth.

5.10 Network Coding as Packet Loss Mitigation Strategy

As was mentioned at the end of chapter 4, we are searching for a packet loss mitigation strategy in order to improve state consistency. In the rest of this section we will describe why intra-flow network coding would be particularly suited as a packet loss mitigation strategy.

5.10.1 Network Coding and UDP

Katti, Muriel and Crowcroft [77] performs inter-flow network coding and show that UDP is particularly well suited to network coding.

In the wireless testbed that they conducted their tests, they used a full wireless 802.11g stack with time-slotted MAC. Firstly they found that network coding improves the throughput beyond the originally predicted coding gain for the scenario shown in Figure 5.3. They suggest that this is due to the more efficient use of the time-slots allocated (by the MAC protocol) to the router: instead of the router using two time slots (such as is show in Figure 5.3 on the left), the router only uses one time slot. As the time-slotted MAC protocol attempts to allocate the time-slots evenly, there is less delay at the router for a multiple time slots, as all three nodes then send in the same number of time-slots.

This effect was the most prominent with UDP flows as each transmission corresponds to one symbol being sent. In TCP flows, the segmentation of packets (according to the flow control algorithm) as well as acknowledgments results in multiple transmission needed to send a single symbol. Therefore the router still requires multiple sending windows and do not see the same gains as UDP.

Based on Katti's results and chapter 4 results showing UDP's better performance under loss conditions, we have decided to implement our network coding strategy by extending the existing UDP implementation. We will refer to our network coding implementation as UDPNC.

5.11 Other Applications of Network Coding

Network coding has been proposed for a wide variety of applications: NC for content distribution in network LTE cells [56], ad-hoc wireless multimedia distribution between mobile devices [117], peer-to-peer streaming [94], data dissemination in V2V networks [84] and network coded TCP for satellite networks [50]. However, we will only dis-

cuss two example applications, network coding in multi-layer video streaming and network coding in freeview video. These applications closely resembles our VE application.

5.11.1 Network Coding in Multi-layer Video Streaming

Many video streaming platforms use a multi-layered video streams to provide video at multiple bitrates (also known as Dynamic Adaptive Streaming over HTTP (DASH)), allowing end devices to choose their required quality and capable speed. If the connection of a device degrades (due to any type of channel fading), it can request lower bit-rates to compensate, viewing in lower quality but keeping the stream current / live. Gheorghiu, Lima, Toledo, Barros and Médard [62] investigates using network coding to improve transmission over lossy Wi-Fi links. Transmission occurs over a typical content distribution network: from video source, through multiple relay servers to multiple video sinks.

Both the video source and relays generate network coded packets using network coding. Each segment of the video stream, containing multiple layers are sent as a generation. A generation first needs to complete before the next generation can be sent. Sending procedure works as follows: firstly, the video source sends the base layer (lowest quality), then each successive higher layer get encoded with layers below it. These stacked encoded layers ensures that lower layers can be decoded first and does not need to wait for all of the higher layers to be received. Note that in normal RLNC, all packets are coded together and the entire generation is needed to decode all the packets. Secondly, relays are also allowed to generate a limited network coded redundant packets from incoming packets as long as the coded packets contain lower or similar video layer information, thus keeping the coded packets immediately decodable at the receivers. Finally, the source server sends redundant RLNC packets containing information from all the layers. As soon as all the sinks confirm that the video is received at their required bandwidths, the source can move on to the next segment.

Using the ns-2 network simulator [10], they show that their network coding strategy maintains video stream throughput even up to 70% loss for the base layer and up to 30% for the higher layers. The same video stream without coding starts degrading under 10% loss with skipped frames and generations. They prove that network coding is useful for such streaming applications and their layer sensitive coding improves higher layer throughput even above traditional

RLNC.

This work is very similar to ours in three ways. Firstly, the application is a video stream, which requires the latest state as soon as possible similar to our interactive virtual environment. Secondly, RLNC is used for creating redundant packets which can be used for forward error correction. Finally, the effect of loss is analysed and compensated for using RLNC packets.

Two major differences include stream versus interactive environment requirements and simulation platform. For the multi-layer stream described, a small start-up delay is allowed, buffering the stream before playback begins. The effect of jitter is also mitigated as the precise frame download time only needs to be before the frame playback time. Buffer is not useful in an interactive VE, because the latest state is required and needs to be consistent with other users of the VE.

The simulation platform used in their study is ns-2 Wi-Fi simulation while ours is Mininet. Ns-2 simulates the full Wi-Fi 802.11 / Ethernet / IP stack which gives a very accurate simulation of reality. Due to the complexity of the VAST software integration, we use Mininet which only simulates networking from the Ethernet layer upwards. Although we apply loss in Mininet, it lacks the Wi-Fi 802.11 interactions. This short-coming will be tackled in future work.

5.11.2 Network Coding in Freeview Video

Zhang, Liu, Chan, and Cheung [124] investigate improvements to freeview video using collaborative network coding. Freeview video allows users to observe a demonstration or event from any angle. This is achieved by capturing the event from discrete angles (called anchors) and generating interpolated viewing angles as necessary. More anchors therefore means a higher quality viewing experience. Anchor data is pulled from the video server using the primary channel, typically a cellular network. Their original contribution is using network coding to provide extra information for neighbouring users in a secondary channel, typically Wi-Fi Direct or similar. They called this system Peer-Assisted Freeview Video (PAFV).

They formulate the problem as a joint optimisation: anchors pulled over the primary channel should minimize distortion of each node's own freeview and of neighbours in its region. Available anchors are RLNC coded and shared on the secondary channel to all neighbours. The combined nature of the RLNC packets allow multiple neighbours to benefit from the single packet. Zhang compare PAFV to two other collaborative approaches, Greedy Anchor Select

(GAS) and Random Anchor Select (RAS). Zhang plots the expected results of freeview video distortion for the entire network for the three approaches. PAFV shows lower distortion overall, even in loss conditions of 5%. PAFV also benefits most from increased nodes in the scenario, as each node creates the more RLNC packets to improve performance. They also showed that the system is practically implementable on Android phones. Using the phones setup, they verified their expected results, showing that PAFV increases freeview video quality. Freeview video is related to VR in the sense that a full 360 degrees video is required for fully immersive VR. Similar to our work, the PAFV uses RLNC to increase usable information at the individual nodes. Although we use the same interface for our regular and redundant network coded packets, the same mindset of fully using the resources available is present. However, like multi-layer video streaming, freeview video allows a small delay before starting playback, which is not useful in our VE.

5.12 Summary

In this chapter we discussed existing packet loss mitigation strategies and erasure codes. We set out the requirements for a packet loss mitigation strategy for state consistency improvement in our VAST distributed Virtual Environment.

We discussed the initial concept of Network Coding in wired and wireless scenarios which can lead to increased throughput. Then we discussed how network coding can also increase reliability by generating redundant symbols, similar to a fountain code. Using network coding for generating redundant symbols does have costs in terms of bandwidth, decoding delay and complexity. We explained that generation sizes could limit the decoding delay and complexity at the cost of bandwidth or vice versa.

We discussed how UDP packets are suitable for use with network coding and how Random Linear Network Codes allow the distributed generation of redundant packets. Finally we discussed two other implementations of interactive applications which also have strict latency and reliability requirements like our state consistency problem. They showed that RLNC could be useful in our application as well.

In the next chapter we will discuss the UDP with Network Coding implementation that we used as a packet loss mitigation strategy.

Chapter 6

Implementing Network Coding in VAST

In the previous chapter, we discussed Network Coding as a possible packet loss mitigation strategy. In this chapter, we will explain how the strategy was implemented as an extension of the UDP network implementation used in Chapter 4. We will refer to our implementation as UDPNC.

In this chapter, we will be using packets or messages when discussing information exchanges of VAST packets. However, the packets are handled as symbols in the network coding paradigm. Therefore if we refer to packets being coded together, we are using the packet as a whole as a symbol, and encoding it with another symbol (whole packet).

6.1 Chapter layout

In section 6.2, we will discuss the network setup which enables generating redundant packets. In section 6.3, we will describe how our network coding strategy fits in the inter-flow and intra-flow paradigms described in section 5.8.5. Before we get into the details of the encoding and decoding processes we will give a visual explanation of our network coding strategy in section 6.4.

In sections 6.5 and 6.6, we will describe our how our implementation performs encoding and decoding in order to provide and use the redundant packets.

Finally in section 6.7, we discuss how our implementation achieves the design requirements and considerations that were outlined in section 5.5.

6.2 Network Coding Network Setup

The network setup (shown in Figure 6.1) for network coding is similar to what we used for the TCP and UDP tests with the addition of the coding host connected to the OpenFlow switch with a high bandwidth, lossless link, like an Ethernet LAN cable.

The POX controller and OpenFlow switch together with the coding host, is our implementation of a coding node. We need a coding host as most SDN switches are OpenFlow enabled, and currently OpenFlow only supports store-and-forward instead of compute-and-forward as we need in our application. Szabo et al [114] presents *Network Coding as a Service* in which they modify the behaviour of a router to enable compute-and-forward operations. In contrast, our design requires only minimal changes to the existing network: we add a flow, using the POX controller, for all incoming UDP packet to be sent to the coding host (with IP address 10.0.0.254) in addition to their intended destinations. We thereby duplicate all of the UDP traffic to the coding host. The coding host decides which packets will be coded together.

6.3 Inter-flow and Intra-flow NC for Interactive Applications

As we explained in section 5.8.5, network coding (NC) can either improve throughput with inter-flow NC or reliability with intra-flow NC. However, neither description completely fits our requirements of a packet loss mitigation strategy in a VE.

The packet loss mitigation strategy needs to have low latency. In chapter 2 we discussed the state consistency problem and how the state consistency problem arises from the need of the users to *interact*. User experience is strongly dependent on the virtual environment being consistent and responding as expected. The user experience and the state consistency requirements give rise to the stringent latency requirements of a VE implementation. Furthermore, in chapter 4 we presented the degenerative effects of network delay on state consistency. The following assumption allows us to assure the minimum latency of our packet loss mitigation strategy:

Only the *latest* update is important, and not the chain of updates before it.

In VAST, the update packets are sent as small stand-alone packets and not delta-updates, therefore this assumption is valid. By focusing on only the latest packet and discarding late packets, we can

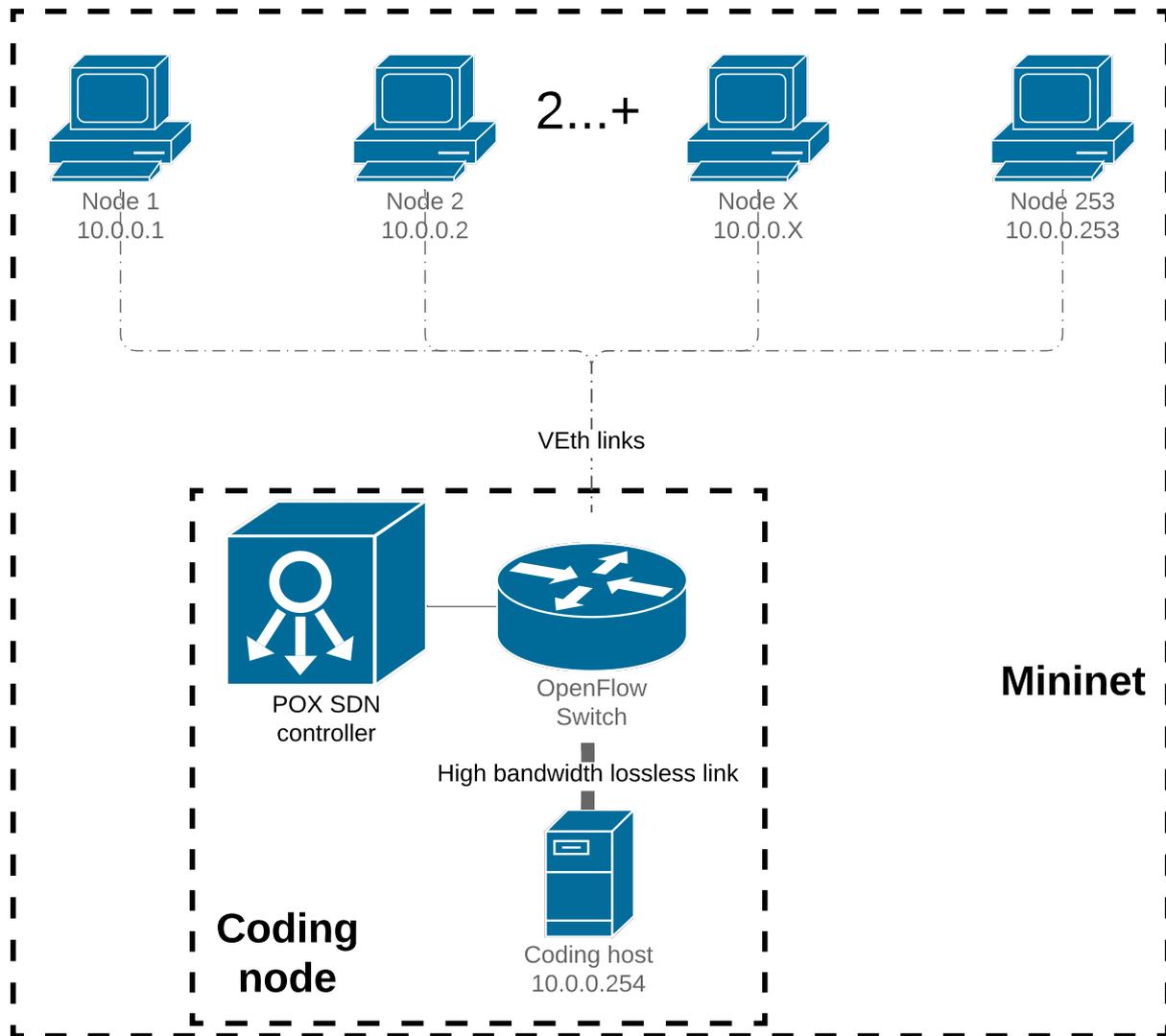


Figure 6.1: Network coding system setup.

reduce the amount of processing delay and have the latest packet available to the VAST library as soon as possible.

From previous work [109], it was explained that intra-flow NC encodes packets from the same packet stream. This would inevitably mean that earlier and later packets in the stream are coded together. Intra-flow NC decoding of these packets under packet loss conditions would cause delay, as was explained in section 5.9.2. Section 5.9.2 also explains how using generations can limit the delay, but we want the minimum delay for the packet loss mitigation strategy. Therefore due to decoding delay we cannot use the pure intra-flow coding approach.

From [109] we see that inter-flow NC is primarily used in order to increase the bandwidth efficiency by combining packets. In contrast, to our goal is to improve reliability under packet loss conditions, potentially by increasing redundant bandwidth. Furthermore, if inter-flow coded or uncoded packets should get lost in transmission, multiple nodes will be unable to decode the received packets, thereby increasing the experienced packet loss and reducing the state consistency.

In order to create the ideal packet loss mitigation strategy for VE applications, we propose a combination of intra-flow and inter-flow coding for *interactive applications*: we combine the *latest* packets of different flows and generate coded redundant packets. Coded redundant packets are sent to nodes in order to recover the packets that were lost in transmission.

A unique benefit of this combination of inter-flow and intra-flow NC is redundancy efficiency: a single generated encoded packet is beneficial to multiple receiving nodes. Consider the following scenario:

If node A sends a packet x_1 and node B sends x_2 , the resulting encoded packet would be $(x_1 + x_2)$. The uncoded packets are sent to their destinations and the encoded packet is sent to both A and B. From the same encoded packet, node A can recover x_2 and node B can recover x_1 . In order to decode, each node needs to keep a packet pool of recently sent and received packets.

The encoded packet is therefore useful to both node A and node B, similar to the throughput gain of original inter-flow NC.

As far as we know, combining inter-flow and intra-flow coding in this way for interactive applications is a unique contribution to literature.

6.4 Network Coding Strategy

As was mentioned in the previous section, packet pools are required to decode the incoming NC packets. Using the example above, if a NC packet $(x_1 + x_2)$ is received, either x_1 or x_2 is needed in order to obtain the other packet. At A, the previously sent packet x_1 , stored in the packet pool, can be used to calculate $x_2 = (x_1 + x_2) - x_1$.

Saving sent and received packets in a packet pool will increase the memory footprint of the VAST application on the node. However, as only the most recent packets from nodes are kept, the packet pool would require limited memory. For this reason, we frequently clear the packet pool during operation.

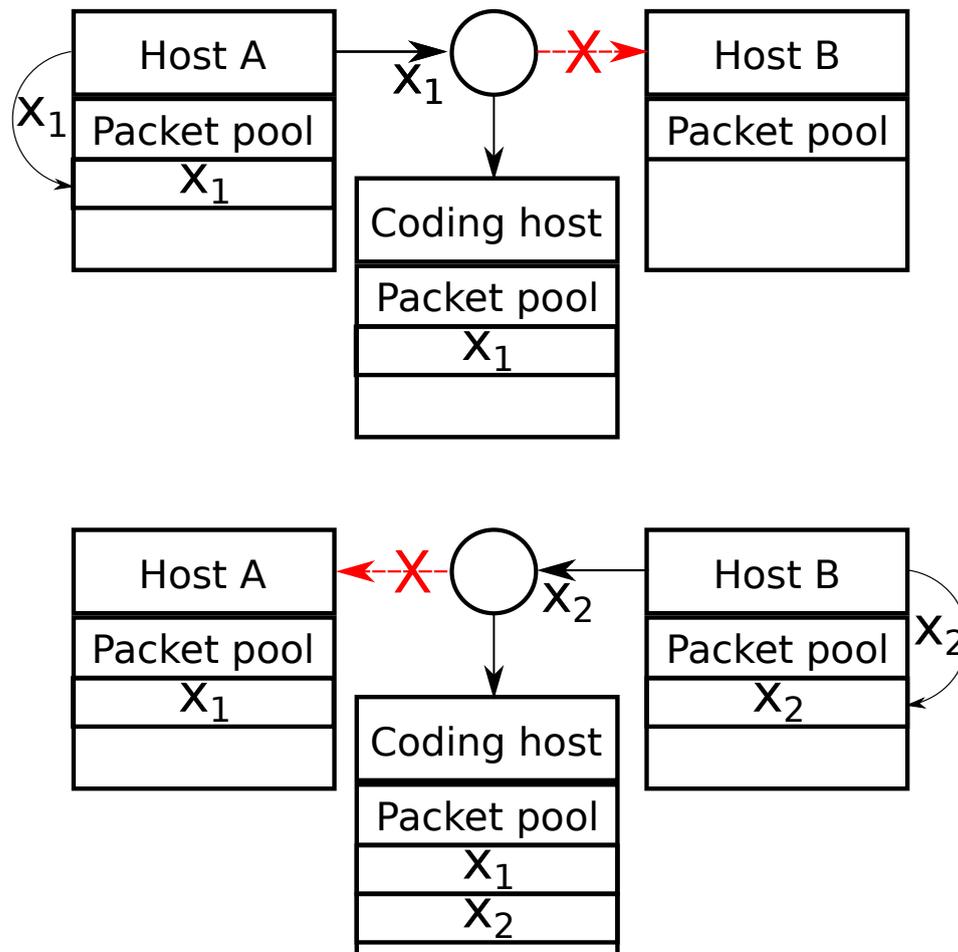


Figure 6.2: Filling packet pools. The circles indicate the network switch or wireless router. The red arrows with a cross indicate a failed transmission. All sent packets are also added to the local packet pool. All UDP packets on the network are also routed to the coding host.

Figure 6.2 shows how the packet pools at the different nodes are filled. All sent packets are stored in the node's local packet pool to be used for decoding later.

The node update steps are not synchronised (because they are distributed), so packets x_1 and x_2 are sent whenever the update step happens. In this way the sequence of x_1 and x_2 is not fixed, but the messages received at the coding host are always from the *latest* update step of each node.

All UDP transmissions are copied to the coding host where they are processed sequentially and as soon as possible. The coding host places all received packets in its packet pool for redundant packet generation. We assume the loss on the router to coding host link is negligibly small because it is placed close to the router and is

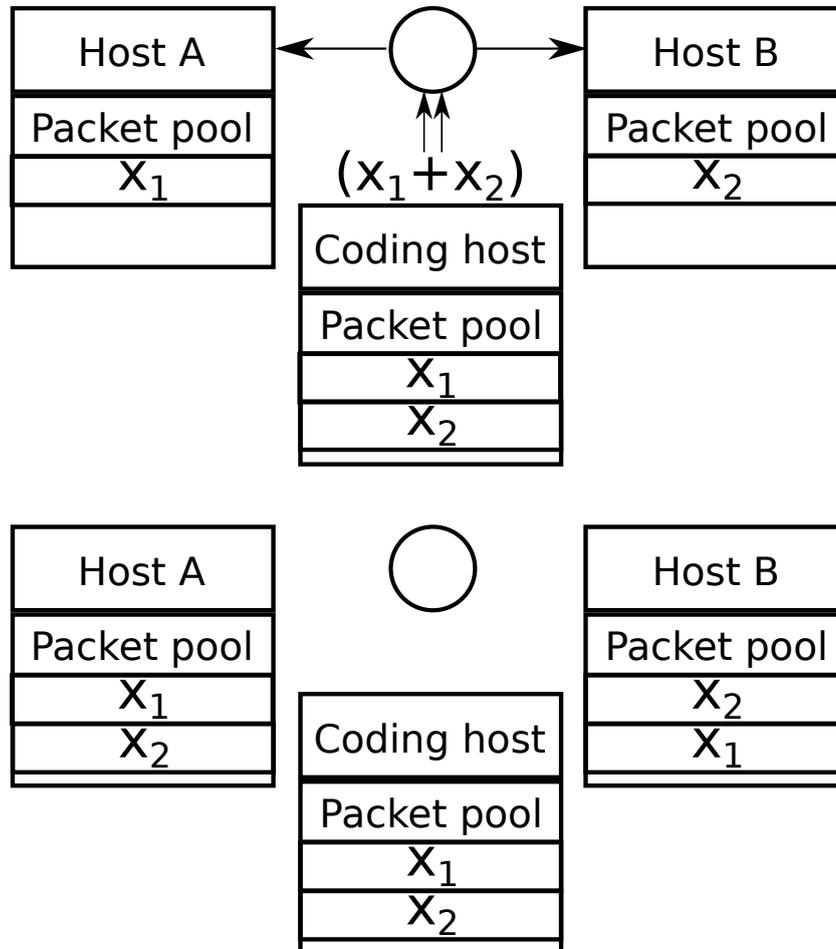


Figure 6.3: Decoding with packet pools. The circles indicate the network switch or wireless router. Packet $(x_1 + x_2)$ is generated by the coding host as a redundant packet. It is forwarded via the router to both recipients that can possibly decode it. Using the stored packets, the other packet can be decoded.

connected with a Ethernet LAN cable. Therefore we do not expect packets to be lost between the router and coding host, even if the transmission fails over the other link.

Packet loss on *both* x_1 and x_2 transmissions is the *worst* case scenario and only typically happens at very high packet loss rates.

Figure 6.3 shows the coding host generating a redundant packet and sending it to the hosts that could possibly decode it. Note that the coding host has no insight into the reception state of the hosts, it simply encodes packets together that are available.

In summary, in addition to the normal VE and network operations, the following is added:

- Sent packets are added to a local packet pool.

Offsets	Octet	1				2				3				4																			
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	start				ordering (byte)				gen_size				generation (byte)				enc_packet_count (byte)															
4	32	packetsize												padding		end																	
8	64	pkt_ids (packetid_t)																															
12	96	from ID (id_t)																															
16	128	to_addr (Vast::IPAddr)																															
20	160																																
24	192																																
28	224																																
32	256	checksum												data[0...*] char																			
36	288																																

Figure 6.4: The UDPNC Header. The shown data types are defined as follows: `byte:unsigned` 1 byte, `packetid_t`: unsigned 8 bytes, `id_t`: unsigned 8 bytes, `Vast::IPAddr`: 4 byte IP address, 2 byte port number, and 2 padding bytes, total of 8 bytes. The checksum field is an unsigned 4 byte integer. The data field is a variable length field based on the packet size field.

- All UDP packets are copied to the coding host.
- The coding host creates redundant packets and sends them on the network.
- Hosts receive and can use redundant packets to recover missing packets.

Note that by switching off the coding host, UDPNC performs almost exactly like the UDP implementation. The differences will be explained in the following sections.

6.5 Encoding Process

In this section, we will discuss all of the components that were implemented in order to generate and deliver redundant packets.

6.5.1 UDPNC Header

The first problem to be addressed is identifying NC packets. This is solved by the UDPNC implementation adding a UDPNC header to all of the VAST packets that it needs to dispatch. Figure 6.4 shows the layout of the UDPNC header.

The information contained in the header helps the coding host determine which packets can be encoded together while still be decodable at the receiving hosts.

The first 4 bits in the start field identifies the UDPNC header, specifically if the value `0xD` is read, it indicates the start of a UDPNC packet.

In our application we use a generation size of one in order to eliminate the decoding delay. Furthermore, as the packets encoded together form a unique generation (i.e. no other packets will ever form part of the generation again), we do not currently use the generation field. Even though the generation and generation size fields are not currently used, they are included to make the UDPNC header extensible.

The encoded packet count field shows how many packets are coded together. In our application this field is either one (uncoded) or two (encoded), but in general RLNC could allow many more packets to be coded together.

The packet size field specifies how many bytes are contained in the payload. The packet size field is large enough to describe any valid VAST message.

The padding field can also be used later if required, but currently ensures that the UDPNC header size is a multiple of 4 bytes allowing efficient memory access on 32 and 64-bit systems.

The 4 bits in the end field, is equal to `0xE`, confirms that the information contained between the start and end bits are indeed a UDPNC header. It is unlikely, though not impossible, that another UDP packet would have the start and end bits in exactly the correct place. If a packet is wrongfully decoded as a UDPNC packet, it is likely that the packet size would not match up with the received size of the packet. It would then be labeled as a corrupted packet and be discarded.

There can be multiple entries in the packet IDs, from IDs and to addresses, depending on the encoded packet count. For each encoded packet, there should be a packet ID, from ID and to address corresponding to that packet.

The checksum field is only present in encoded packets, otherwise it is set to zero. The CRC-32 checksum of each of the individual packets is computed and summed together. At the decoding side the decoding can be checked by also computing the sum of CRC-32 checks of the packet pool packet and decoded packet.

Finally, the data field contains the message from VAST that should be dispatched.

Note that this UDPNC header is added to each packet in addition to the normal UDP header, leading to a higher overhead for UDPNC communication. However, the UDPNC header is required for the successful encoding and decoding operations.

6.5.2 Coding Host

The coding host performs the task of RLNC recoding, i.e. network coding performed inside the network. The coding procedure described in Figure 6.5, is repeated every time a new packet arrives. It follows these steps:

1. Search for packets from different hosts by checking the *from* IDs in the UDPNC headers. If no two such packets are available, stop and wait for a new packet. Only packets from different hosts (and therefore different flows) are allowed to be coded together because we are only interested in the *latest* updates. Two packets from the same host will inevitably contain at least one outdated packet.
2. If found, create a new UDPNC header to encapsulate the encoded packet. Add the packet IDs, from IDs and to addresses to the new encoded packet header. Use the RLNC recoder to generate an encoded packet. In terms of the example shown in Figure 6.3, the coding host will wait until it receives both x_1 and x_2 . It will then have two packets from *different* source hosts and can encode them together.
3. Copy the packet IDs, from IDs and to addresses to the outgoing encoded packet so that it can be processed at the end nodes correctly.
4. Generate the checksum. In order to ensure that the packets are decoded correctly at the end nodes, a checksum of the two individual packets are computed, summed and added to the packet.
5. Send the same packet in multiple unicasts to the individual packets' intended destinations (based on the *to address* in the UDPNC header).

It is a design decision that the number of packets in the coding host's packet pool remain as small as possible. The packet pool is implemented using a C++ standard map with $O(\log N)$ look-up complexity. Furthermore, as we are checking for packets to encode together, at worst we need to transverse the entire map ($O(N)$). Therefore keeping the packet pool size small, will reduce the processing delay at the coding host.

Because of the considerations above, packets will not be kept for multiple encodes, but rather as soon as a packet (x_1+x_2) is produced, x_1 or x_2 is discarded.

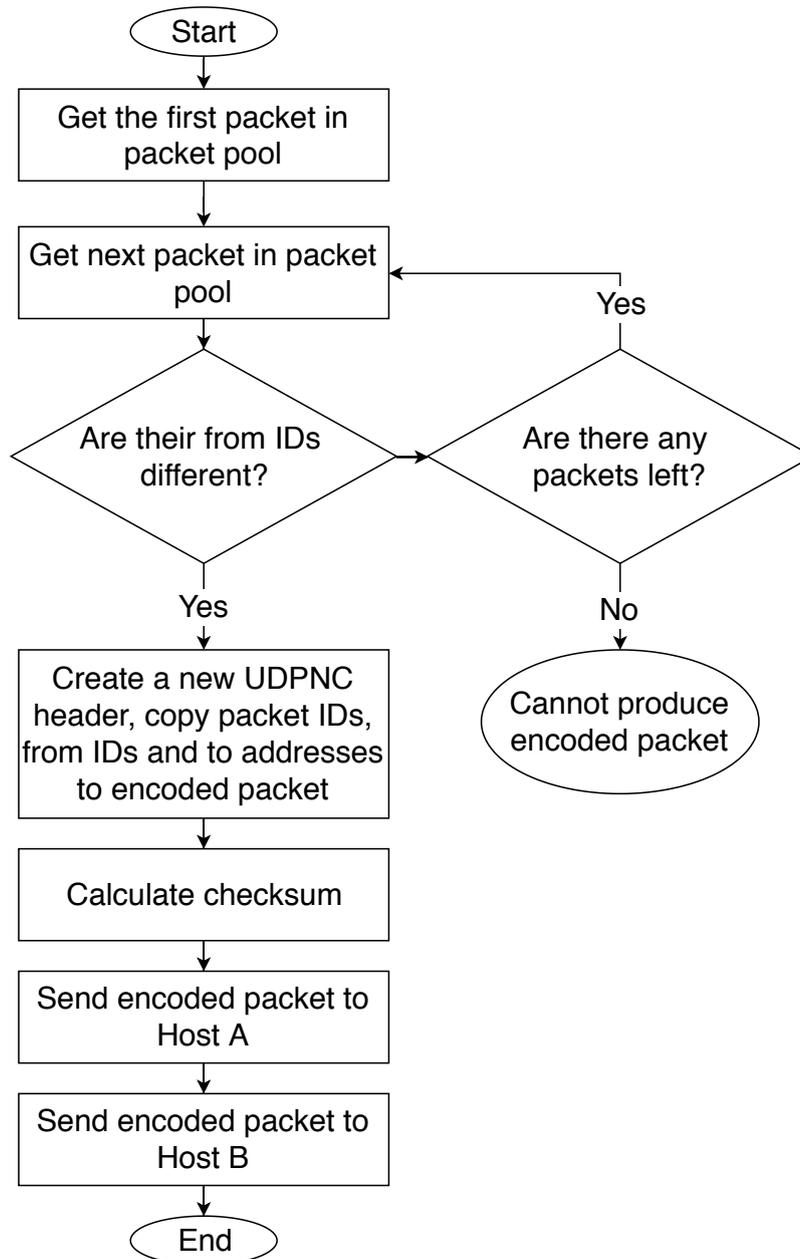


Figure 6.5: Coding process

Furthermore, as encoded packets are latency sensitive, it is likely that x_1 and x_2 will be stale when the next packet arrives for coding.

Note that only RLNC recoding happens and no source encoding is used in order to demonstrate the efficacy of *network* coding.

6.5.2.1 Multiple Coded Packets Trade-offs

In our application we only combine two packets and discard the source packets used in order to limit latency and keep packet pool sizes small. However, it is worth considering the benefits and drawbacks of alternative strategies.

The first option would be to send different combinations of pairwise coded packets. This would increase the bandwidth required, but yields more redundant packets to aid loss recovery. However, packet flows will have to be considered as well, because even though a receiver could be able to decode the packet, the decoded packet's destination could be a different receiver.

The second option is sending a second round of network coded packets. As network coding can be used as a fountain code, potentially any number of packets can be combined to form a coded packet. However, at the receiving ends the combined packet should still be decodable. Therefore the coding host should combine the packets in such a way that the receivers have a high probability of decoding.

Consider the following scenario: Host A should receive x_1 , but the transmission failed. During the first round of encoded packets, it overhears $x_2 + x_3$. Then in the next round, it receives $x_1 + x_2 + x_3$. Host A is now able to extract $x_1 = (x_1 + x_2 + x_3) - (x_2 + x_3)$. Note that this would incur a significant latency as it has to wait for both the redundancy symbols to arrive. However, if the coding host transmits at a fast enough rate, the state consistency can be recovered before the next uncoded transmission for host A arrives. As with any erasure code, sending more redundant symbols require more bandwidth and in this case could also increased latency. Furthermore, this strategy requires wireless overhearing (receiving a wireless transmission without being the intended recipient). Overhearing has been shown in literature to significantly increase network coding opportunities ([77], [81], [120]).

Increased redundancy could be very useful for packets that require higher reliability, for example monetary transactions or once-off environmental changes such as a door being opened.

6.6 Decoding Process

The UDPNC implementation needs to be able to process uncoded and encoded UDPNC packets. In order to accommodate this, we divide the receiver into two pipelines for coded and uncoded packets. After the processing of packets are done in both pipelines, they need

to be reassembled into one consistent stream to be processed by the VAST library.

6.6.1 Uncoded pipeline

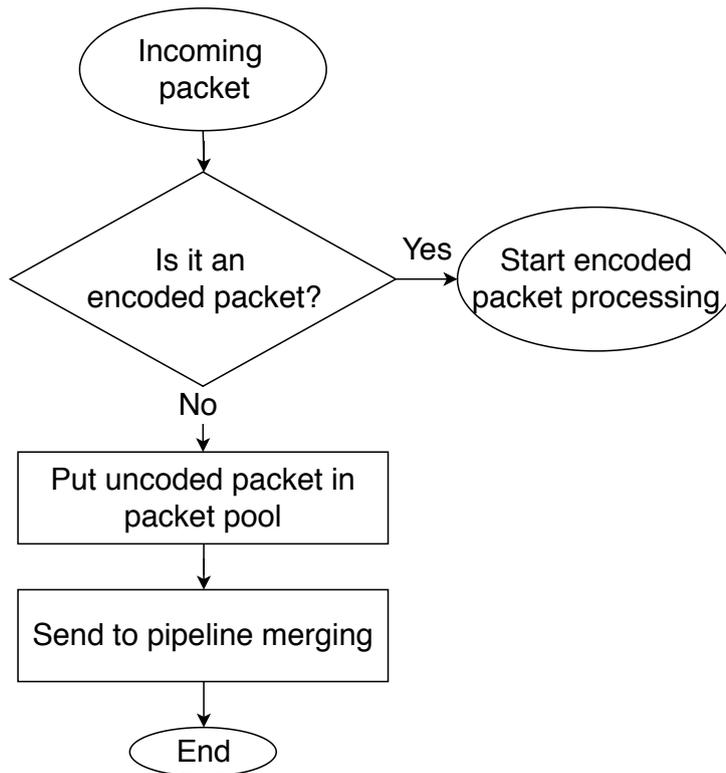


Figure 6.6: Uncoded packet pipeline

The uncoded pipeline is shown in Figure 6.6: all packets arrive at a node from the network and are processed by the UDP socket. The UDPNC header's encoded packet-count is checked and packets with an encoded count larger than one are passed to the coded pipeline. In order to maximise the decoding opportunities, the uncoded packet is also placed in the local packet pool. Finally, the uncoded packet is sent to the merging pipeline.

6.6.2 Coded pipeline

The coded pipeline shown in Figure 6.7 is more complicated than the uncoded pipeline as it needs to perform decoding as well. Note that this pipeline starts when an encoded packet is received.

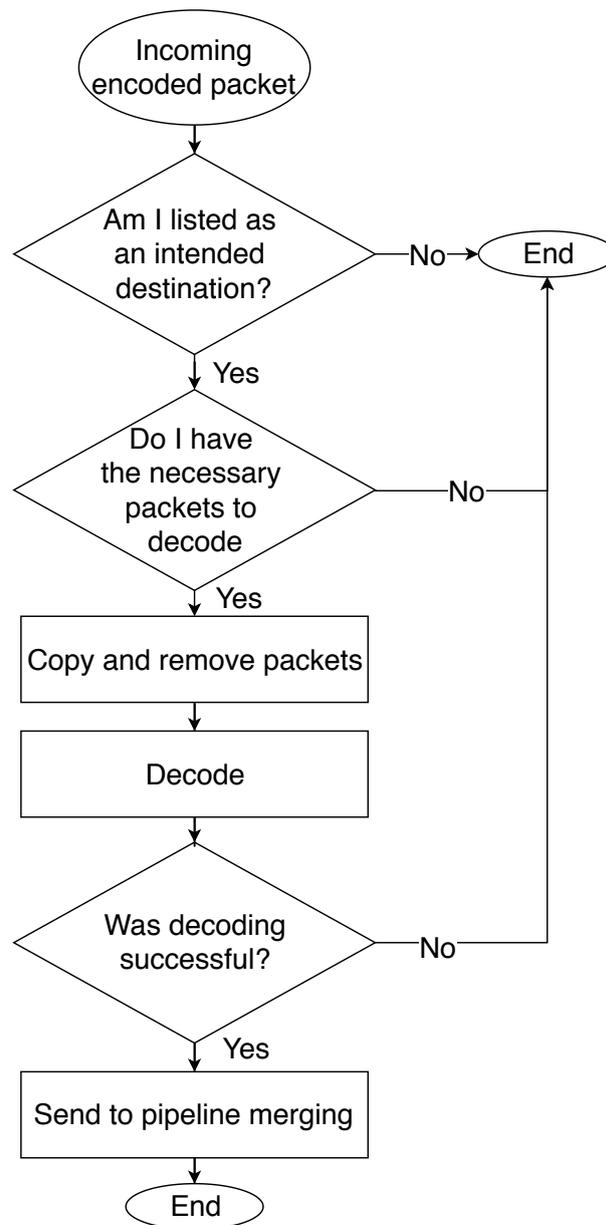


Figure 6.7: Coded packet pipeline

1. Firstly, the to addresses in the UDPNC header is checked to ensure that this node is an intended recipient. If not, the decoding processes is aborted.
2. The packet pool is then checked to see if the necessary packets are available (from unicast sends or receives). This look-up operation is performed in $O(\log N)$ time as the packet pool is

sorted according to packet ID.

3. If the packets are available, they are copied to the decoder and removed from the packet pool.
4. The decoder then uses the packets to attempt a decode. Note that if the packets are encoded such that the linear equations cannot be solved, the decoder will report a failed decode. This happens because the coefficients of the linear equations are chosen randomly, which has a probability to generate linearly dependent equations.
5. If the decode was successful, the decoded message is passed to the merging pipeline.

Note that there are many paths that would abort the decoding process. This is done to ensure that the processing delay is as small as possible.

In RLNC, the set of linear equations need to be solved in order to recover the original packets. Each node has $(\alpha_1x_1 + \alpha_2x_2)$ and either x_1 or x_2 . Solving this set of equations is trivial, except when the randomly chosen $\alpha_i = 0$, making the extraction of packet i impossible. The simple equations ensure that decoding delay will be minimised. In our tests we recorded very few such unsolvable scenarios.

6.6.3 Merging pipelines

These two pipelines can run independently from each other and could lead to race conditions when sending the uncoded or decoded packets to the VAST library. We describe two such scenarios which could adversely affect the state consistency:

6.6.3.1 Double update

If an update packet is received both through the uncoded and coded pipeline, it is possible that the update packet can be passed to the VAST library twice. This would result in the server or client experiencing the other client's position to remain fixed for two updates. Furthermore, because the updates are processed sequentially, the extra (duplicate) update could delay the correct next update and thereby decrease the state consistency.

6.6.3.2 Out-of-order update

The processing delay of the coded pipeline is longer than the uncoded pipeline due to the extra decoding steps. In the worst-case

scenario, the decoding of a lost uncoded packet could take so long that a newer uncoded update has already arrived. If the decoded packet (now representing a stale update) is passed to the VAST library, the decoded update will move the client to an older, incorrect position. This will cause serious harm to the state consistency, not because of the induced delay as above, but rather because extra state inconsistency is introduced.

In both these cases, correct ordering would prevent duplicate or stale updates to be applied. In the next section we will describe how our implementation provides ordering.

6.6.4 Producer-Consumer Ordering

As was mentioned above, the pipelines are not necessarily synchronised and could cause a race condition on passing the updates to the VAST library. We address this problem by using a Producer-Consumer pattern for multi-process synchronisation and ordering.

In the producer-consumer pattern typically a queue is used to transport data. The producer adds data to the queue when it is available, while the consumer removes it as needed. The queue uses mutual exclusion to allow either the producer or consumer access. That is, the producer cannot add data while the consumer is removing data. This pattern separates the producer thread from the consumer thread, allowing them to perform actions independently. However, this pattern introduces new edge cases that need to be addressed: what happens when the queue is empty, i.e. the consumer has nothing to process and what happens when the queue is full and there is no space for the producer to add a packet. The decisions regarding the edge cases as well as how the producer-consumer pattern is used will be explained below.

In our implementation there are two producers: the uncoded and coded pipelines. Both pipelines add packets to the queue. This means that the mutual exclusion needs to be passed between both pipelines and the consumer.

Firstly, if the queue is full, the producers will have to wait for the consumer to clear enough space for placing more packets in the queue. This could potentially increase processing delay, but was not observed during our tests.

On the consumer side, we have a thread which simply processes the queue packets as quickly as possible and passes them to the VAST library. If the queue is empty of packets, the consumer waits for a producer to signal that it has placed a packet in the queue.

In using a single thread for the consumer, we can correct the possible reordering of the packets without the fear of race conditions.

The ordering algorithm is shown in Figure 6.8. We will first describe the algorithm and then explain the reasoning behind our decisions.

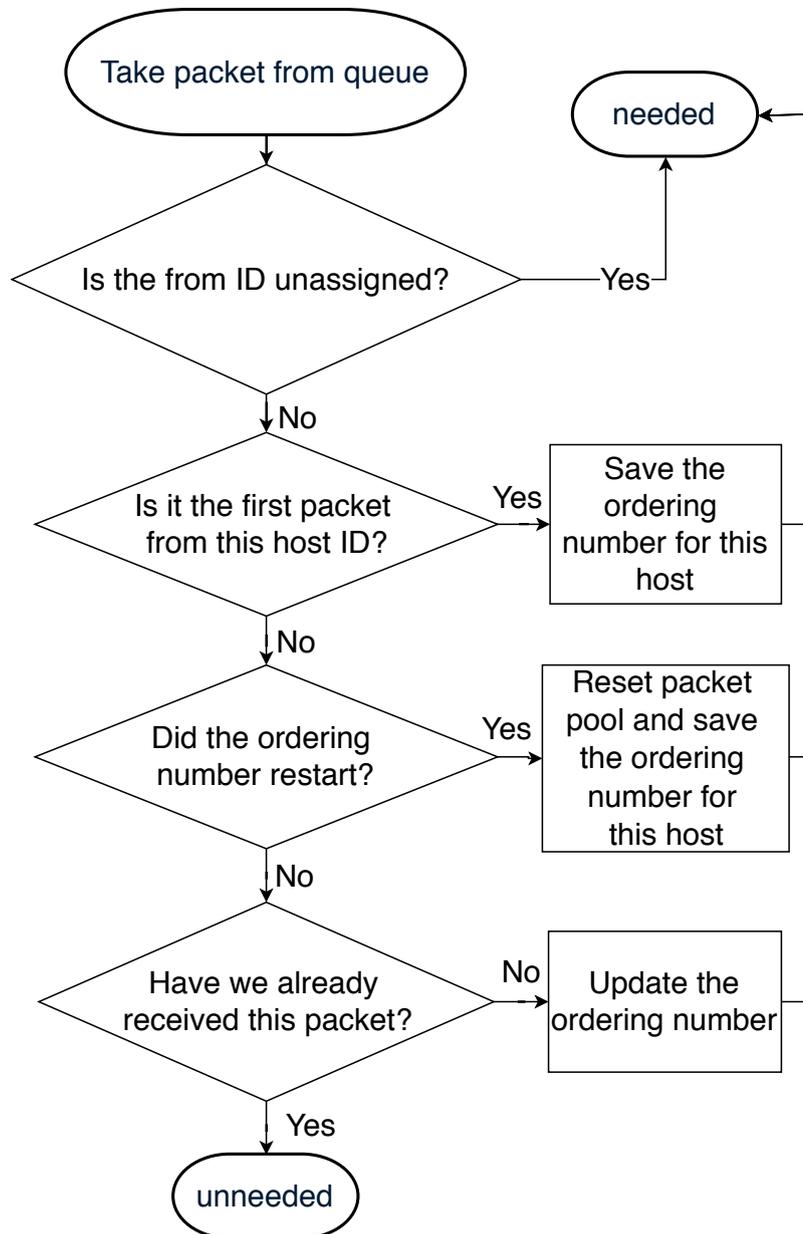


Figure 6.8: Order input

1. After taking a packet from the queue, the from ID in the UDPNC header is checked. When a new node joins the VE, its host ID is not yet assigned, i.e. from ID is zero. These packets are sent

to the Matcher to request an ID. Without a host ID, no further communication can happen in the VAST overlay, therefore these packets always need to be allowed. The coding host will also never encode these packets as it cannot determine if they are from different sources, thus there will be no duplicates of these packets.

2. If the node has never heard from this host ID before, it can safely allow the packet and store the ordering number. If a duplicate of this packet arrives, the host ID will no longer be new and the packet will be ordered according to the next steps.
3. The ordering number is represented by a byte and will thus overflow after 255, we need to check if the ordering number restarted. If it did restart, we clear the packet pool and save the new, restarted ordering number. This is the only case where we accept a lower ordering number, eg. 1 after 255.
4. If none of the above applies, the packet is either a new packet (will have a higher ordering number), a duplicate or an old packet. We only accept packets with higher ordering numbers than what we have previously received, discarding the rest. If we accept this packet, we save its ordering number and pass it to the VAST library.
5. After the process is completed, the consumer takes the next packet from the queue.

The ordering number is specific to each pair of host IDs, that is, each host increments separate ordering numbers for each flow.

It was decided to clear the packet pool every 255 packets, in order to keep the packet pool small, limiting processing delay in the map structure. This also corresponds to the choice of using only one byte for the ordering number. The integer overflow provides a convenient solution for scheduling the packet pool clearing. As we are only interested in the latest updates, it is inconceivable that a sequence of more than 255 packets would need to be kept if we are only interested in the latest one.

If many packets in the sequence are missing, the ordering number of the received packets could jump, for example from 252 to 3, with packets 253 - 255 and 0 - 2 lost in transmission. In order to compensate for this, we detect any jump between 240 to 10 as a packet rollover. This wide range could cause a small temporary reordering, but will be recovered as soon as the ordering number

6.7. FULFILLING DESIGN REQUIREMENTS AND CONSIDERATIONS **143**

becomes larger than 10. The choice of rollover bounds could be investigated in future work. Ideally the bounds should be adapted according to the packet loss rate.

If any of the paths in Figure 6.8 are followed to the *needed* endpoint, the packet is passed to the VAST library. Otherwise the packet is discarded.

6.7 Fulfilling Design Requirements and Considerations

In this section, we discuss how our UDPNC implementation fulfills the design requirements and considerations mentioned in section 5.5.

6.7.1 Network Delay Minimise

UDPNC is not a retransmission strategy, but rather an erasure code. The redundant packets are available soon after the original uncoded packet would have been. The coding host is also optimised to minimise the processing delay by limiting packet pool size as well as sending packets as soon as they are available. The encoded packets are sent to only those nodes that can possibly decode them so as not to flood the network with undecodable packets, limiting transmission and receiver queues.

6.7.2 Processing Delay Minimised

At the decoding nodes the minimum number of decoding operations are performed and packet pool sizes are kept small in order to minimise the processing delay. Furthermore, a generation size of one is used to prevent extra waiting for packets later in the generation. The producer-consumer pattern allows the packets from the network to be received in the producer threads, but further ordering and passing to VAST library happens in the consumer thread. The consumer processes packets as quickly as possible.

6.7.3 Distributed Mitigation Strategy

RLNC recoding allows the coefficients to be assigned at each node without centralised coordination. The ordering numbers need to be communicated between nodes, but are included in the UDPNC header and also do not need centralised coordination. Redundant

packets are generated based on which packets are available at the coding host which functions independently.

6.7.4 Keep MAC / Physical Layers Unchanged

The UDPNC protocol is built on top of the UDP, below the VAST application layer. Therefore no changes are required to the physical, MAC and Internet (IP) layers. The nodes do need to be able to decode the UDPNC packet, so the UDP network implementation would not be forwards compatible with UDPNC. Therefore all nodes in the network would need to run UDPNC. Other UDP packets, such as DHCP and NTP would not be affected by the UDPNC application.

6.7.5 Software Defined Network Integration

Using the coding host allows UDPNC to operate on an SDN network without custom router software. The POX controller can be configured to perform other tasks as well as the UDPNC routing. The coding host can also be deployed as a Virtual Network Function in the network.

6.7.6 Keep Original VAST Implementation

By implementing UDPNC as a VAST network implementation, similar to the TCP and UDP implementations, the VAST library's functionality remains exactly the same, allowing comparisons to previous work.

6.8 Summary

In this chapter, we have discussed the concept and detail of our UDPNC implementation. We explained the networking setup that we used for UDPNC tests as well as our inter-flow redundant NC strategy. We illustrated how the interactions between the hosts, router and coding host works.

We then explained the detail of the encoding at the coding host and the decoding and ordering at the nodes. Finally, we have explained how our implementation satisfies our design requirements set out in Chapter 5.

In the next chapter, we will evaluate our proposed packet loss mitigation strategy and compare it to the UDP baseline results.

Chapter 7

Results of Using Network Coding as Packet Loss Mitigation Strategy

In this chapter we will show the results of using UDP with network coding as the network implementation for VAST. We will use the results from the UDP tests in chapter 4 as a comparison. We repeated the experiments in chapter 4 to get comparable results for UDPNC. Experimental setup and simulation parameter details are repeated here for the benefit of the reader.

7.1 Network Coding Network Setup

The network setup (shown in Figure 7.1) for network coding is similar to what we used for the TCP and UDP tests with the addition of the coding host connected to the OpenFlow switch with a high bandwidth, lossless link, like an Ethernet LAN cable.

7.2 Simulation Parameters

The same simulation parameters (shown in Table 7.1) was used as in prior tests. Simulation parameters are based on previous work by Hu et al. [70].

7.3 VAST Components

In order to ease discussion of the simulation parameters and results, we remind the reader of two VAST roles as was discussed in section 2.8.

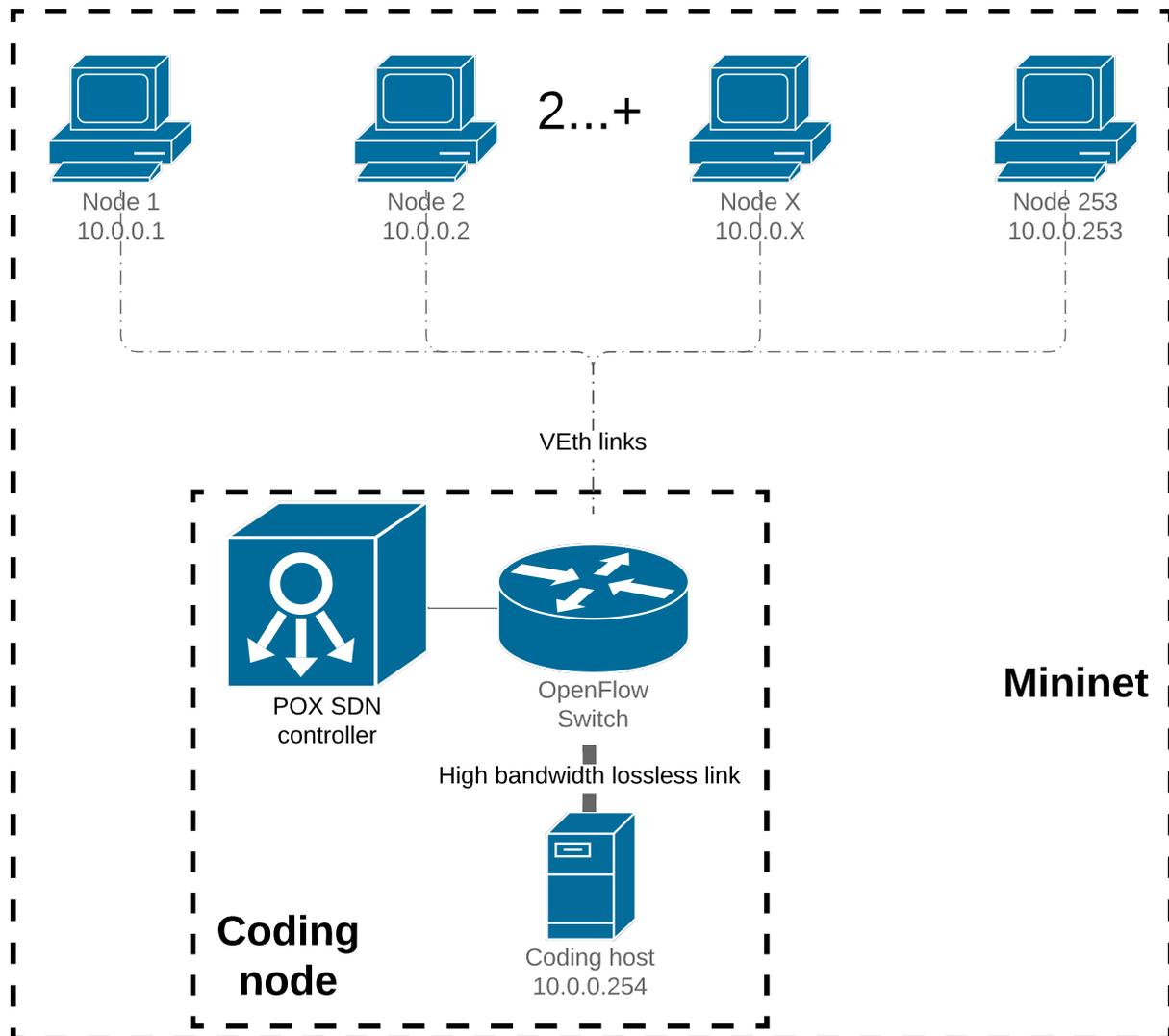


Figure 7.1: Network coding system setup.

The *Matcher* is a super-peer responsible for managing the spatial subscriptions and forwarding spatial publications to the correct subscribers.

The *Client* is a normal peer in the VE. Clients (representing users) move around in the VE and can subscribe to areas they are interested in. Note that a node can host both a matcher and a client.

Table 7.1: General simulation parameters

Parameters	
World dimension (units)	768 x 768
AOI radius	195
Step duration	100 ms
Simulation setup steps	$10 * K + 1000$
Total Simulation steps	5000
Join rate	1/s or 1/10 steps
Speed (units / step)	3
Movement model	Uniformly random
Number of matchers	1
Iterations	20

7.4 Nominal test

We perform a test under nominal network conditions (no packet loss, no network delay, no bandwidth limitations). We perform this test to compare the specific implementations, before evaluating their performance under adverse conditions.

7.4.1 Experiment Setup

Each network implementation was used to run a simulation of 5000 steps with 50 nodes in the network. No limitations on bandwidth and no adverse network effects were applied. This setup is the ideal case for any network application. The nodes are started by the Mininet topology script as specified by the join rate. The first node will be promoted to the matcher role, therefore having both the matcher and a client. All other nodes will only have a client that connects to the first node's matcher. When the simulation is finished (5000 steps completed), the Mininet topology script will stop all the VAST processes and hosts. The consistency logs will be analysed and results written to file. The mean of each run will be computed from these result files and plotted using boxplots.

7.4.2 Results

Table 7.2 shows a near 100% topology consistency for all network implementations.

TCP has the lowest drift distance of 2.06 VE units. Surprisingly UDPNC has a drift distance of 2.24 VE units, which is 17% lower than the drift distance of UDP.

CHAPTER 7. RESULTS OF USING NETWORK CODING AS PACKET LOSS
MITIGATION STRATEGY

Table 7.2: Medians and 90% confidence interval of state consistency and network metrics for nominal test.

	TCP	UDP	UDPNC
Topology Consistency	100.00 ± 0.00	99.99 ± 0.00	100.00 ± 0.00
Drift distance	2.06 ± 0.01	2.71 ± 0.01	2.24 ± 0.02
Latency	114.81 ± 0.25	142.46 ± 0.36	123.65 ± 0.57
NIC send	7.54 ± 0.00	6.48 ± 0.00	7.25 ± 0.00
NIC receive	7.54 ± 0.00	6.48 ± 0.00	15.77 ± 0.00

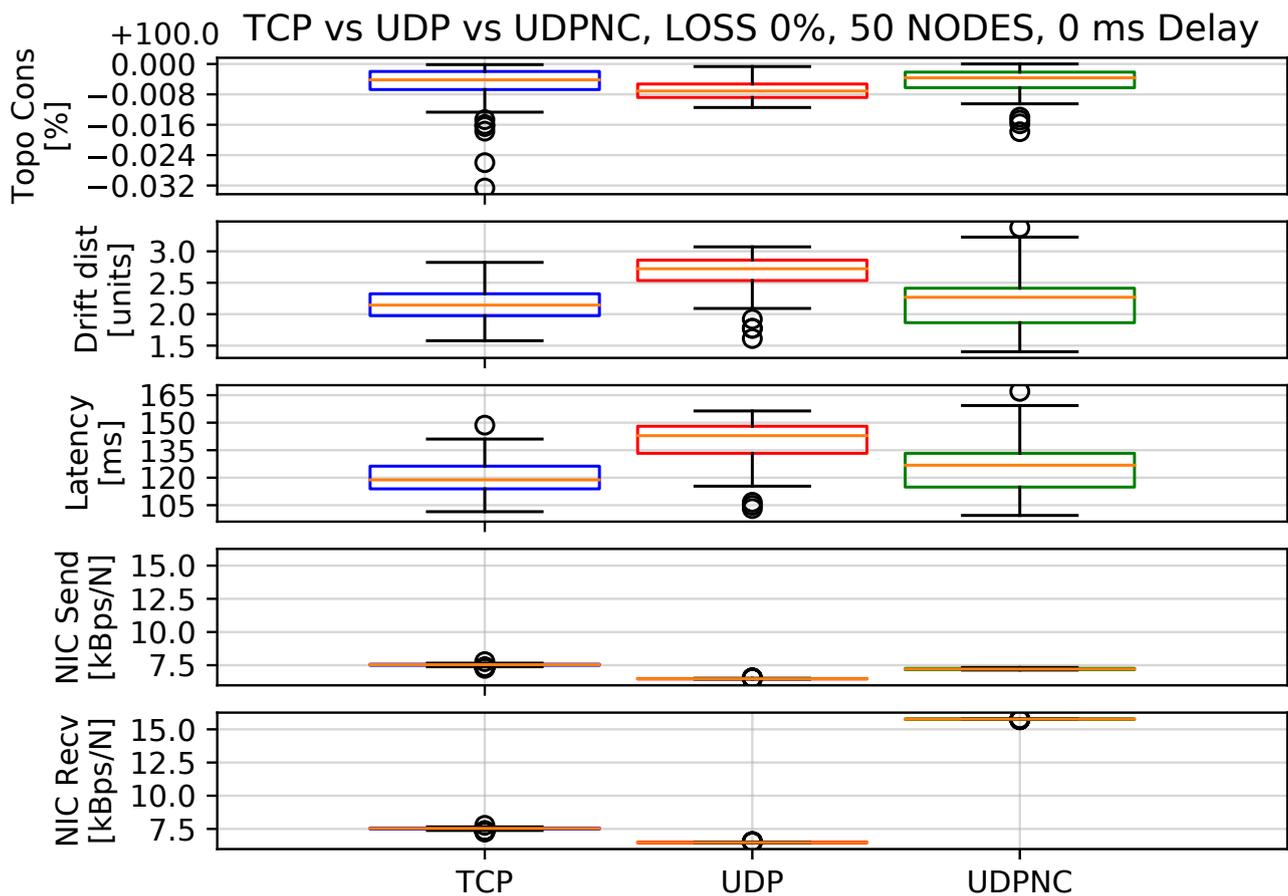


Figure 7.2: Comparison of TCP, UDP and UDPNC under nominal conditions. Aggregate topology consistency, normalised drift distance, average latency, and normalised NIC send and receive bandwidths are shown for TCP (blue), UDP (red) and UDPNC (green) experiments. Black circles indicate outliers.

In increasing order, TCP, UDPNC and UDP have 114 ms, 123 ms, and 142 ms latencies respectively. The drift distance trend matches the latency.

In terms of send bandwidth, TCP and UDPNC are higher (7.54 kBps and 7.25 kBps per node) with UDP the lowest (6.48 kBps per node). Receive bandwidth for UDPNC is by far the highest (15.77 kBps per node).

Figure 7.2 shows that TCP state consistency performance is the highest and UDP the lowest. Although UDPNC median drift distance is lower than UDP, the whiskers show that UDPNC drift distance is very variable.

7.4.3 Discussion

Firstly, it is clear that none of the network implementations add enough processing delay as to degrade VE state. Although UDP median drift distance is 2.71 VE units compared to the 2.06 VE units of TCP, both network implementations still limits the average drift distance to less than one step distance (3 VE units).

The excellent performance of TCP can be explained by its differences compared to UDP and UDPNC. TCP is multi-threaded as it uses a processing thread for each connection. This means that incoming packets are handled in a multi-threaded way, reducing the length of each connection's processing queue. That is, multiple short queues instead of a single long queue. Note that latency in VAST is measured by the application and therefore include the queuing and processing delay. The network delay for all the implementation are similar (as controlled by Mininet), but the reported application latency can vary depending on the specific network protocol stack and implementation.

UDP has only one thread for processing incoming packets, which leads to longer processing times. The effect can be seen in the longer latency and larger drift distance when compared to TCP.

UDPNC is a two threaded process. One thread handles receiving and decoding of packets, adding the received and decoded packets in a queue. The other thread orders packets, dispatching them to VAST as quickly as possible. This leads to a slightly more efficient processing pipeline as the first thread can continue reading more packets while the second is performing processing tasks. However, this does introduce multi-threaded problems such as resource access contentions or race conditions. These resource access contentions are part of the reason for the high variability in latency and drift distance of UDPNC: the handing over of resource access is a kernel task and is therefore dependent on the kernel and processor

load, which is inherently variable in a multiprocess environment. Another part of the variability is the decoding process and different arrival times of uncoded and network coded packets: as the packets to be encoded are first sent to the coding host, the network coded packets will arrive later than the uncoded packets. If the network coded packet is needed for recovery, the specific packet will register a larger experienced latency. Packet loss is random, and therefore the utilisation of uncoded and network coded packets are variable, leading to variable latencies.

Compared with UDP, UDPNC has a higher NIC send bandwidth. The higher bandwidth is due to larger overhead in UDPNC. The overhead of UDPNC contains the required fields for correctly decoding packets, including the coding vector, ordering and packet IDs. All packets on the network are sent with a custom UDPNC packet header to enable the coding process in the coding host and the decoding process in the receiving nodes.

The extra NIC receive bandwidth is generated by the coding host, combining sent packets from the nodes to generate redundant NC packets. These packets are sent to every node that can decode them in multi-unicasts, leading to much larger total NIC receive bytes.

7.5 Why we only look at UDP vs UDPNC

We predominantly use UDP as a comparison for UDPNC in this chapter. In chapter 4 it was seen that UDP is better suited to adverse network conditions, therefore we will use it as a baseline for VE performance. Furthermore, UDPNC is based on the UDP implementation, allowing accurate comparisons.

7.6 Packet Loss Mitigation Evaluation and Adverse Network Conditions Testing

In the following section we will investigate the packet loss mitigation capabilities of UDPNC by repeating the packet loss test from Chapter 4 under 0 - 70% packet loss rates. We will also verify the packet loss test result obtained from Mininet simulations by evaluating VAST performance on a real network using physical hosts.

In the later sections, we will repeat the network delay and bandwidth limitations tests. We will also investigate the scalability of UDPNC by evaluating the state consistency under increasing node numbers.

In the subsequent delay and scaling tests we will observe the state consistency under 10% packet loss in order to represent the lossy conditions of a wireless network. We repeat the arguments for the packet loss rate decision here for clarity:

According to Sheshadri and Koutsonikolas [110], using 802.11n Wi-Fi without MAC layer retransmissions can result in packet loss rates varying between 10% - 80% depending on the Modulation and Coding Scheme (MCS) used. It is not always possible to disable MAC retransmissions as the MAC layer could be implemented in hardware and therefore in practice MAC retransmissions would be enabled by default. However, in Mininet we are not able to emulate the mechanics of the MAC retransmissions and therefore we perform simulations as if they are disabled. Our simulation therefore probably assumes too high packet loss rates for wireless and do not account for retransmission delay. However, in the delay tests, the effect of possible retransmission delay is evaluated.

Using 10% packet loss (the lower bound on packet loss) in the delay and scaling tests aligns with our purpose of investigating the effect of adverse network conditions on the VE state consistency, potentially using wireless, lossy communication.

In the bandwidth tests we will not use the 10% packet loss as the bandwidth limitations are already very restrictive and other adverse network conditions would prevent the VAST overlay to operate at all.

7.7 Packet Loss Tests

As we did in section 4.11, we will perform tests at different packet loss percentages to determine how effective UDPNC is in mitigating packet loss.

7.7.1 Experiment Setup

In this experiment, a normal VAST simulation is run as was specified in section 7.4.1. A Mininet topology was used to specify packet loss rate. Packet loss is only applied after all of the VAST clients have joined the VE and the VAST overlay has reached steady state.

Note that packet loss is applied on downstream links only. That is, we apply the packet loss on data moving from the switch to the nodes and keep the data flow originating from the nodes destined for the switch lossless. Applying packet loss on only one of the links is comparable to applying half the value of packet loss on both links. We therefore do not alter the simulation significantly by applying packet loss only to the downstream links. UDPNC is only able to

Table 7.3: Packet loss test simulation parameters

Packet loss	0-70%
Number of nodes	50
Delay	0 ms

recover packets on the downstream links. We run the current tests with downstream packet loss, as we did in Chapter 4.

Mininet applies packet loss using the command line utility `tc`. The documentation for `tc`, specifically its network conditions emulator can be found here [7]. In this network emulator, each processed packet is assigned a random probability to be lost in transmission, according to the packet loss percentage required.

In addition to the simulation parameters given in Table 7.1, this specific test uses the simulation parameters given in Table 7.3. We chose a maximum packet loss rate of 70%, similar to Gheorghiu et al's paper. This is a reasonable upper limit as transmission becomes very difficult for any protocol above this value.

7.7.2 Single run analysis: UDPNC

As we did before in Section 4.11.4 for UDP, we will first look at a single example run of UDPNC. Figure 7.3 shows the measured values during a typical run at 10% loss.

The bottom graph in the figure represents the redundant packets received from the coding host. Adding the average of the NIC sent bytes and the redundant receive bytes results in the NIC received bytes. Redundant packets also experience packet loss on the downstream link, and therefore the redundant bandwidth also shows variability.

Point c) indicates the end of the simulation setup time, when packet loss rate is applied. An anomaly can be seen at 130000 ms and also at 340000 ms, marked as points a) and b) on Figure 7.3. This anomaly can be seen both before and after packet loss is applied, in the drift distance, NIC receive bytes and latency graphs. Upon closer investigations of program logs, we suspect this is caused by the order number roll over in the ordering process (see section 6.6.4). During this roll over, the packet pool is cleared, preventing the decoding of NC packets until they become available again in the next timestep and taking some time to clear the packet pool map (linear with respect to packet pool size). Although this roll over process happens frequently during the simulation, each node performs the roll overs at different times. However, at certain intervals, these packet pool clearing happens at the same time, causing more

Table 7.4: Median of average topology consistencies with 90% confidence interval when increasing packet loss.

	UDP	UDPNC	\pm %
0.0 %	99.99 \pm 0.00	100.00 \pm 0.00	0.00 %
1.0 %	99.98 \pm 0.00	99.99 \pm 0.00	0.01 %
2.0 %	99.97 \pm 0.00	99.98 \pm 0.00	0.01 %
5.0 %	99.93 \pm 0.00	99.96 \pm 0.00	0.03 %
10.0 %	99.84 \pm 0.00	99.89 \pm 0.00	0.05 %
15.0 %	99.69 \pm 0.00	99.82 \pm 0.00	0.13 %
20.0 %	99.51 \pm 0.00	99.70 \pm 0.00	0.19 %
30.0 %	98.80 \pm 0.00	99.27 \pm 0.00	0.48 %
40.0 %	97.23 \pm 0.00	98.27 \pm 0.00	1.07 %
50.0 %	93.81 \pm 0.00	95.87 \pm 0.01	2.20 %
60.0 %	86.60 \pm 0.01	91.01 \pm 0.02	5.10 %
70.0 %	71.04 \pm 0.05	77.50 \pm 0.06	9.09 %

nodes to experience a processing delay simultaneously. As this happens on multiple nodes, this degrades the state consistency for a short while. Despite the anomaly it can be seen that the recovery is fairly quick and does not affect the overall average drift distance too greatly.

Avoiding the simultaneous packet pool clearing could be achieved by adding a random interval after the roll over to clear the packet pool or to allow a larger roll over interval. However, the latter strategy could increase the packet pool map lookup times, therefore the former strategy is suggested for future work.

7.7.3 Aggregated results

Although topology consistency remains high, from Table 7.4 it can be seen that both UDP and UDPNC struggle to keep the state consistent after 60% packet loss (86% and 91% respectively).

Table 7.6 shows the degradation percentages from the no packet loss case. Even though the UDPNC degradation (2279%) is larger than UDP (1924%), it should be noted that the starting drift distance of UDPNC (2.27 VE units) was lower than UDP (2.72 VE units). Therefore, as UDPNC degrades to UDP drift distance, the relative degradation is more.

In Figure 7.6 and Table 7.5 we can see that UDPNC outperforms UDP over the entire packet loss percentage range. From 60% onwards, both UDP and UDPNC drift distance increases rapidly (with 26.64 and 24.80 VE units at 60% packet loss), with even UDPNC

CHAPTER 7. RESULTS OF USING NETWORK CODING AS PACKET LOSS
MITIGATION STRATEGY

154

MITIGATION STRATEGY

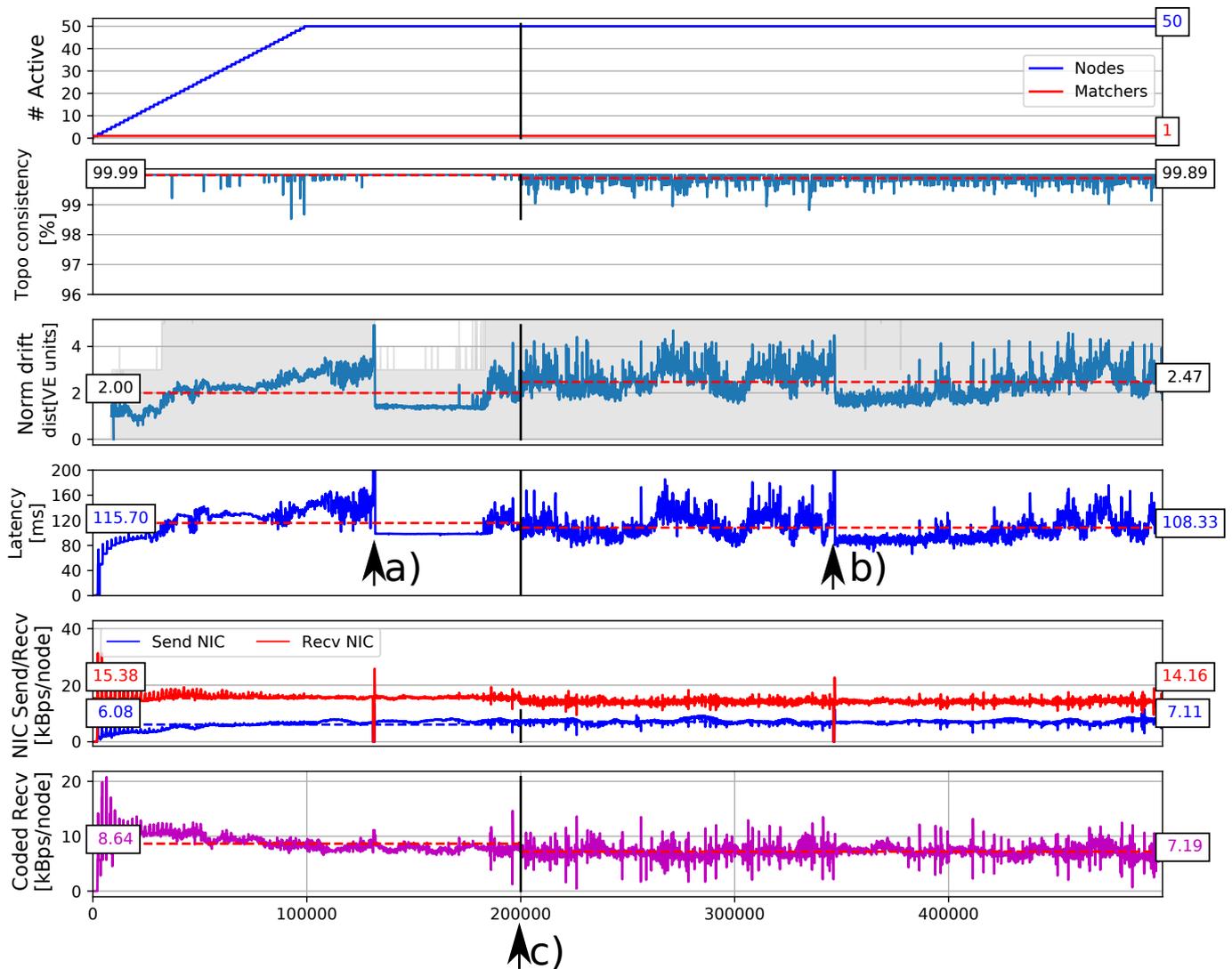


Figure 7.3: Results of a UDPNC run showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase, also marked as point c). Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. In this figure the drift distance plot is scaled to the range of normalised drift distance for readability. Points a) and b) indicate anomalies in drift distance, latency and NIC receive bytes due to packet pool clearing. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. See Figure 7.4 for full range of maximum drift distance.

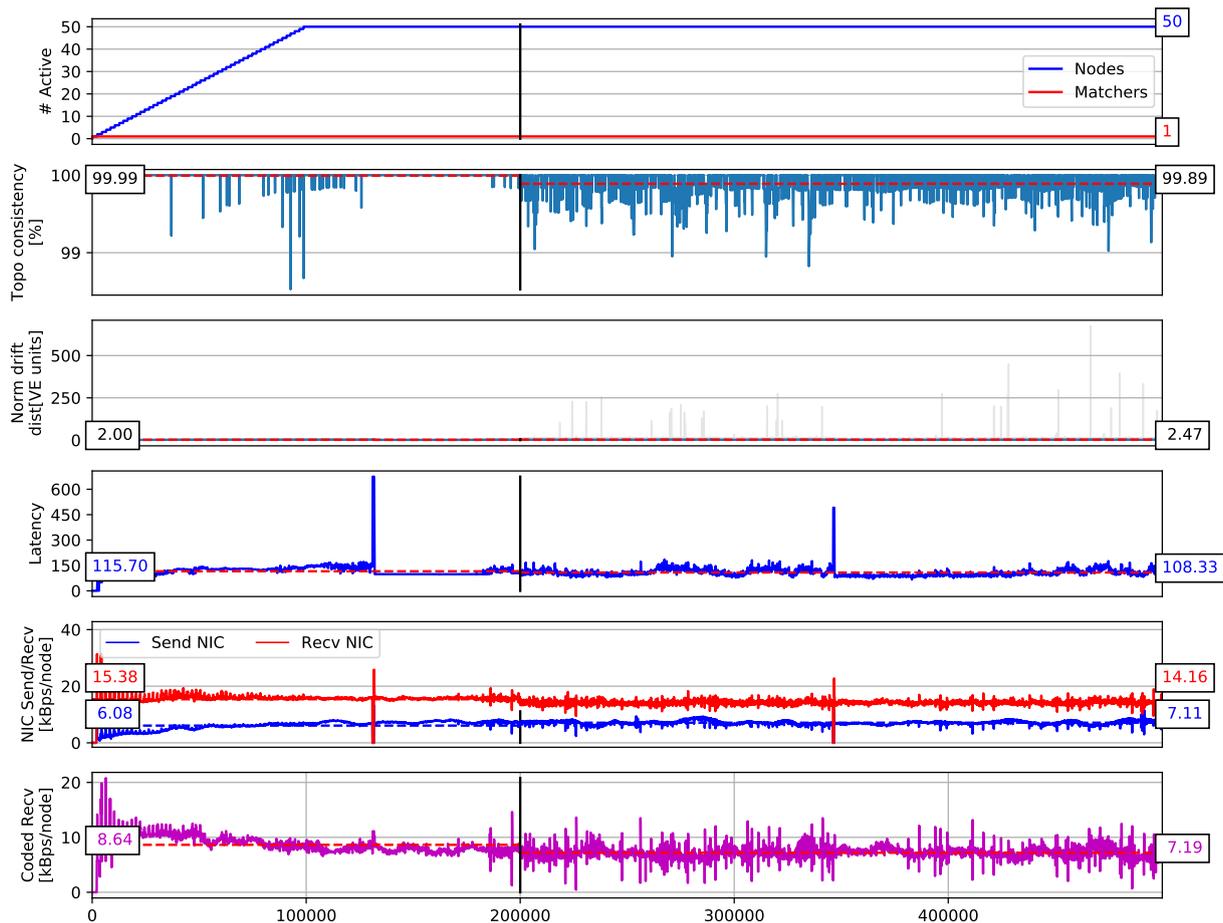


Figure 7.4: Results of a UDPNC run showing topology consistency, drift distance, latency, and send / receive bandwidths over time. The test was run at 10% packet loss. The black line indicates the end of the setup phase. Packet transmissions until the black line are lossless. After the black line, packet loss is applied. The maximum drift distance recorded is shown in gray. Averages of topology consistency, normalised drift distance, send / receive bandwidth and latency under lossless conditions is given on the left. Averages under lossy conditions is given on the right. Drift distance plot now shows the full range of maximum drift distance.

CHAPTER 7. RESULTS OF USING NETWORK CODING AS PACKET LOSS
156 MITIGATION STRATEGY

Table 7.5: Medians of normalized drift distances [VE units] with 90% confidence interval when increasing packet loss.

	UDP	UDPNC	± %
0.0 %	2.72 ± 0.01	2.27 ± 0.01	-16.68 %
1.0 %	2.75 ± 0.01	2.11 ± 0.01	-23.44 %
2.0 %	2.71 ± 0.01	2.15 ± 0.01	-20.70 %
5.0 %	2.87 ± 0.01	2.18 ± 0.02	-23.99 %
10.0 %	3.21 ± 0.00	2.55 ± 0.01	-20.65 %
15.0 %	3.74 ± 0.01	2.72 ± 0.01	-27.37 %
20.0 %	4.06 ± 0.01	3.07 ± 0.01	-24.33 %
30.0 %	5.78 ± 0.01	4.09 ± 0.01	-29.30 %
40.0 %	8.79 ± 0.01	6.55 ± 0.02	-25.48 %
50.0 %	14.73 ± 0.01	12.49 ± 0.03	-15.16 %
60.0 %	26.64 ± 0.04	24.80 ± 0.05	-6.89 %
70.0 %	55.12 ± 0.15	54.00 ± 0.21	-2.04 %

Table 7.6: Medians of normalized drift distances [VE units] with degradation percentages when increasing packet loss.

	UDP	degrade %	UDPNC	degrade %
0.0 %	2.72	0.00 %	2.27	0.00 %
1.0 %	2.75	1.02 %	2.11	-7.17 %
2.0 %	2.71	-0.50 %	2.15	-5.30 %
5.0 %	2.87	5.35 %	2.18	-3.90 %
10.0 %	3.21	17.97 %	2.55	12.35 %
15.0 %	3.74	37.29 %	2.72	19.67 %
20.0 %	4.06	49.14 %	3.07	35.45 %
30.0 %	5.78	112.34 %	4.09	80.17 %
40.0 %	8.79	222.67 %	6.55	188.60 %
50.0 %	14.73	440.84 %	12.49	450.70 %
60.0 %	26.64	878.23 %	24.8	993.15 %
70.0 %	55.12	1924.35 %	54.0	2279.98 %

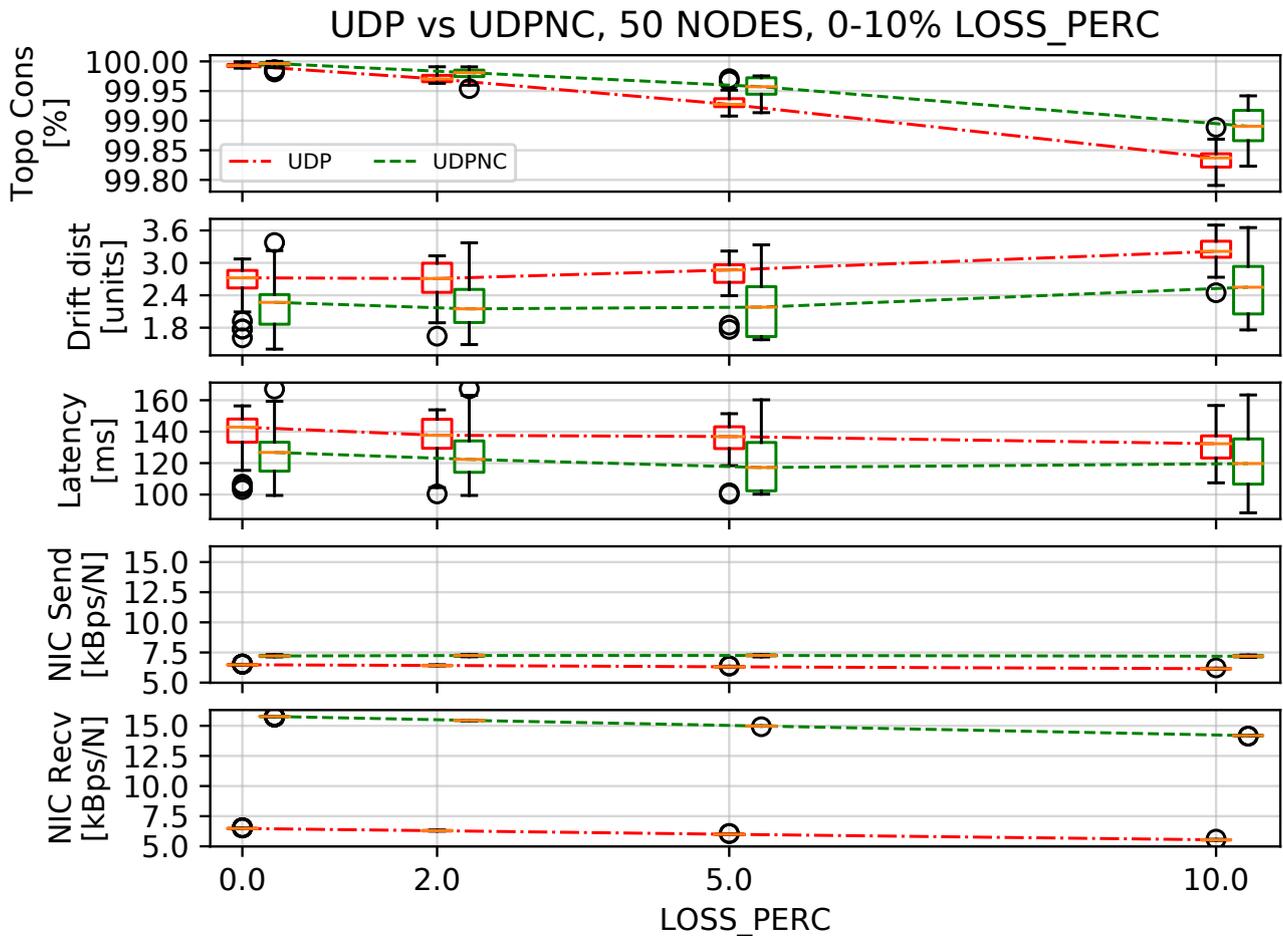


Figure 7.5: Aggregated summary of 20 runs, each with 50 nodes. UDP results shown in red, UDPNC in green. Outliers are indicated with black circles. Only packet loss percentages up to 10% is shown in this graph to facilitate readability. Full results are shown in Tables 7.4, 7.5 as well as Figure 7.6.

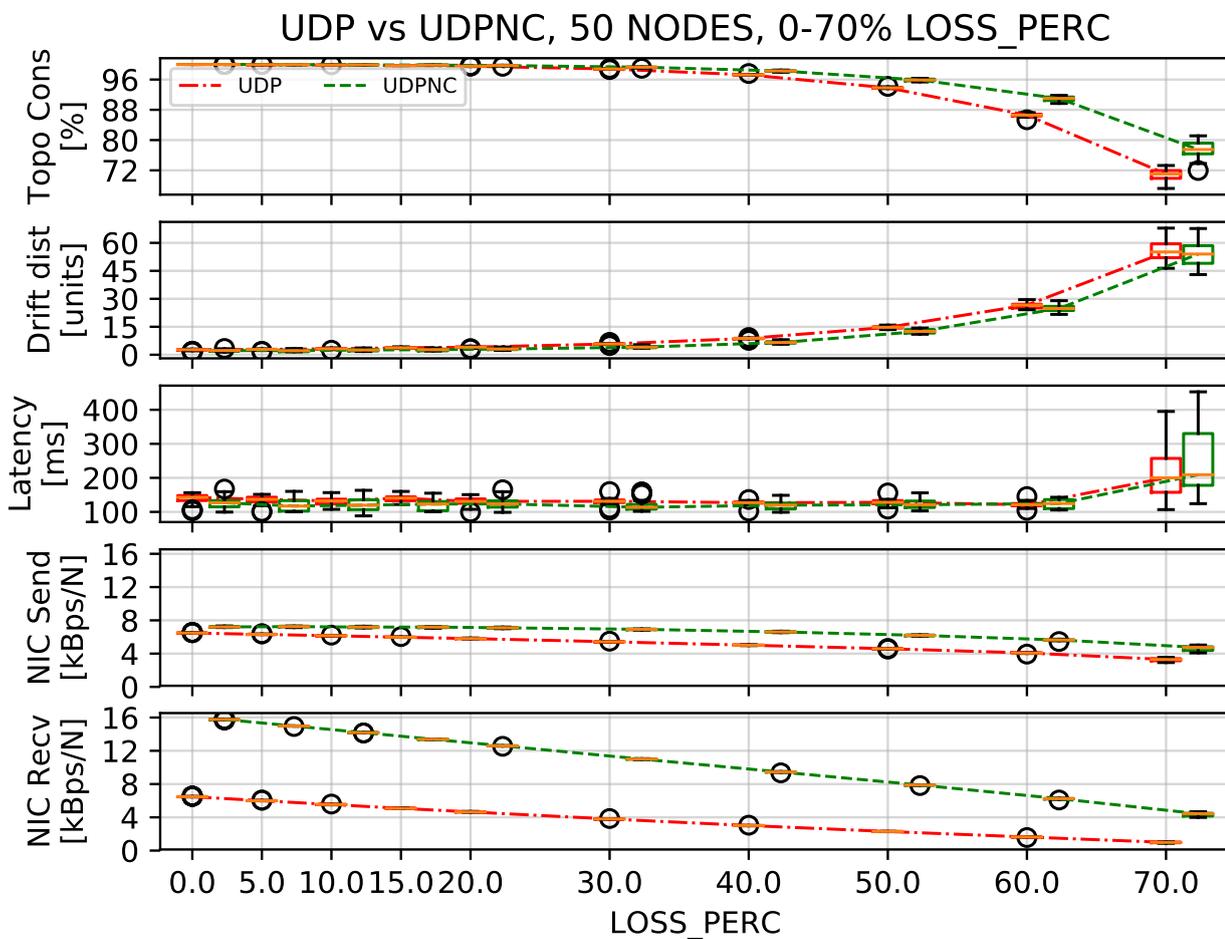


Figure 7.6: Aggregated summary of 20 runs, each with 50 nodes. UDP results shown in red, UDPNC in green. Outliers are indicated with black circles. The entire range of 0-70% packet loss rates is shown.

Table 7.7: Median of normalised NIC receive bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when increasing packet loss.

	UDP	total	UDPNC	total	\pm %
0.0 %	6.48 \pm 0.00	323.89	15.77 \pm 0.00	788.51	143.45 %
1.0 %	6.37 \pm 0.00	318.67	15.61 \pm 0.00	780.52	144.93 %
2.0 %	6.29 \pm 0.00	314.66	15.45 \pm 0.00	772.48	145.49 %
5.0 %	6.00 \pm 0.00	299.84	14.98 \pm 0.00	748.94	149.78 %
10.0 %	5.54 \pm 0.00	276.91	14.19 \pm 0.00	709.26	156.14 %
15.0 %	5.08 \pm 0.00	253.87	13.38 \pm 0.00	669.19	163.59 %
20.0 %	4.65 \pm 0.00	232.31	12.60 \pm 0.00	629.81	171.10 %
30.0 %	3.80 \pm 0.00	189.96	11.00 \pm 0.00	550.12	189.60 %
40.0 %	3.01 \pm 0.00	150.57	9.45 \pm 0.00	472.47	213.80 %
50.0 %	2.29 \pm 0.00	114.7	7.88 \pm 0.00	394.05	243.55 %
60.0 %	1.63 \pm 0.00	81.64	6.25 \pm 0.00	312.37	282.63 %
70.0 %	0.99 \pm 0.00	49.28	4.44 \pm 0.01	221.75	350.00 %

struggling to compensate for high packet loss. Thus UDPNC drift distance converges to the UDP drift distance.

As with previous packet loss tests, a slight downward trend can be seen in the NIC send bytes, indicating a loss of interactivity.

From Figure 7.6 and Table 7.7 it can be seen that UDPNC NIC receive bandwidth is significantly higher than UDP (15.77 kBps versus 6.48 kBps per node).

From Figure 7.5 and Figure 7.6 it can be seen that UDPNC latency is lower than UDP for low packet loss rates. However, at packet loss rates of higher than 60% UDPNC latency increases to equal or more than UDP, as well as exhibiting more variability.

7.7.4 Discussion

UDPNC NIC receive bandwidth is much higher than UDP: 143% higher at no packet loss. The higher receive bandwidth is the cost associated with the network coding redundancy as was explained in section 5.9. This extra bandwidth could limit scalability. However, if the bandwidth is available, UDPNC is a valid packet loss mitigation strategy.

Another noteworthy result is the average latency of UDP versus UDPNC: initially the UDPNC latency is significantly lower than the UDP but increases as packet loss increases. The initial lower latency can be explained by the two-threaded design of the UDPNC implementation. The steady increase in latency can be explained by the extra time that recovered UDPNC packets experience:

1. They need to travel to the coding host to be encoded.
2. The coding host chooses packets from different hosts, so might have to wait until there is a packet available from a different host, especially under high loss conditions.
3. The coded packet then needs to travel to the receiving nodes.
4. Finally, the decoding also adds processing time.

After all of these steps, the recovered packet is passed to VAST, which record the latency. Note that VAST only records the latency of packets that it does receive, therefore UDP does not record overdue packets, but UDPNC can because the overdue packets are recovered.

The most significant improvement of UDPNC over UDP is observed at 30% packet loss, with a 29.3% lower drift distance. The improvement required 189.60% increased receive bandwidth. Although UDPNC drift distance then degrades to UDP performance, topology consistency is still 9.1% higher for UDPNC at 70% packet loss.

7.8 Comparing Mininet Hosts with Physical Hosts

In order to verify our results in the packet loss test from the previous section, we evaluate the performance of UDPNC on a system with a OpenFlow switch, physical hosts and a coding host. The coding host was run as a virtual machine on the same PC that was used as the OpenFlow switch and POX controller, therefore the packet loss and network delay to the coding host from the switch is negligibly small.

However, the OpenFlow switch that could be obtained was a PC running the Open Virtual Switch [12] (OVS) software and only allowed four physical hosts to connect. Therefore we evaluate the performance using four physical hosts. For comparison we also evaluated the performance in Mininet using four Mininet hosts.

7.8.1 Experiment Setup

In order to obtain the Mininet results, we follow the exact same procedure as was specified in Section 7.7.1, except we only use four Mininet hosts instead of 50.

In every aspect possible the physical hosts are identical to the Mininet hosts:

Table 7.8: Mean and standard deviation of 100 round-trip ping times on Mininet and physical network. Results in microseconds.

Link	Mininet		Physical	
	Mean	Std dev	Mean	Std dev
h1 -> h2	43.8	3.89	253.3	37.8
h2 -> h1	42.2	4.00	260.8	35.2
h1 -> coding	42.8	2.82	253.3	37.8
coding -> h2	42.9	3.16	220.4	25.7

- The same VAST library software is also used on both physical hosts and Mininet hosts.
- The same Linux operating system is used on both physical and Mininet hosts.
- The same POX controller is used with both the PC's OVS and Mininet OpenFlow switches.

Therefore the only differences between the physical hosts are the following:

- Physical wires are used to connect to the switch, which could cause a small delay. The round-trip latency was measured and the results are given in Table 7.8.
- The host clocks are synchronised with the Network Time Protocol (NTP).
- Each of the clients run on a separate host and therefore the processing delay would not influence each other. This would be beneficial to the state consistency as processing delay per host is reduced.
- Packet loss rates are applied using the Linux `iptables` [6] filters at the receiving hosts, instead of by Mininet (using `tc`) on network links. `iptables` controls the packet filtering in the Linux kernel, whereas `tc` controls the Linux network scheduler in the kernel. Both utilities allow the probabilistic dropping of packets. However, as `tc` only manipulates outgoing packets and we wanted to drop packets on the downstream link, we decided to use `iptables` to apply packet loss at the receiving hosts. Applying packet loss at the switch was not possible as we did not have control over the internal functioning of OVS.

In addition to the simulation parameters given in Table 7.1, this specific test uses the simulation parameters given in Table 7.9.

Table 7.9: Comparison test simulation parameters.

Packet loss	0-20%
Number of nodes	4

Table 7.10: Median of normalised drift distances [VE units] and 90% confidence interval for UDPNC test when comparing Mininet hosts to physical hosts.

	Mininet	Physical	\pm %
0.0 %	2.44 ± 0.01	1.73 ± 0.04	-29.04 %
10.0 %	2.50 ± 0.01	1.54 ± 0.00	-38.40 %
20.0 %	2.77 ± 0.01	2.47 ± 0.07	-10.71 %

Table 7.11: Median of average latencies [ms] and 90% confidence interval for UDPNC test when comparing Mininet hosts to physical hosts.

	Mininet	Physical	\pm %
0.0 %	105.68 ± 0.28	98.37 ± 0.75	-6.91 %
10.0 %	112.26 ± 0.43	85.93 ± 0.08	-23.45 %
20.0 %	114.09 ± 0.34	119.89 ± 0.56	5.09 %

7.8.2 Results

Table 7.10 shows that UDPNC has a lower drift distance for all measured packet loss percentage values.

Furthermore, Table 7.11 shows that UDPNC on physical hosts has lower latencies for 0 - 10% packet loss (eg. 86 ms versus 112 ms at 10% packet loss). However, at 20% packet loss UDPNC latencies on physical hosts exceed Mininet host latencies (119.89 ms versus 114.09).

From Figure 7.7 we can see that UDPNC mostly performs better on physical hosts, but that there are a large number of outliers in topology consistency and drift distance performance.

7.8.3 Discussion

Surprisingly, UDPNC latencies on physical hosts are lower than on Mininet hosts, contrary to TCP and UDP. Although this is an unexpected result, it can possibly be explained by the multi-threaded processing of UDPNC: by separating the receive functionality from the ordering and VAST library interactions, a lower processing delay could be experienced. Furthermore, because physical hosts use separate CPUs instead of sharing a CPU (as in Mininet), multiple threads can run more efficiently.

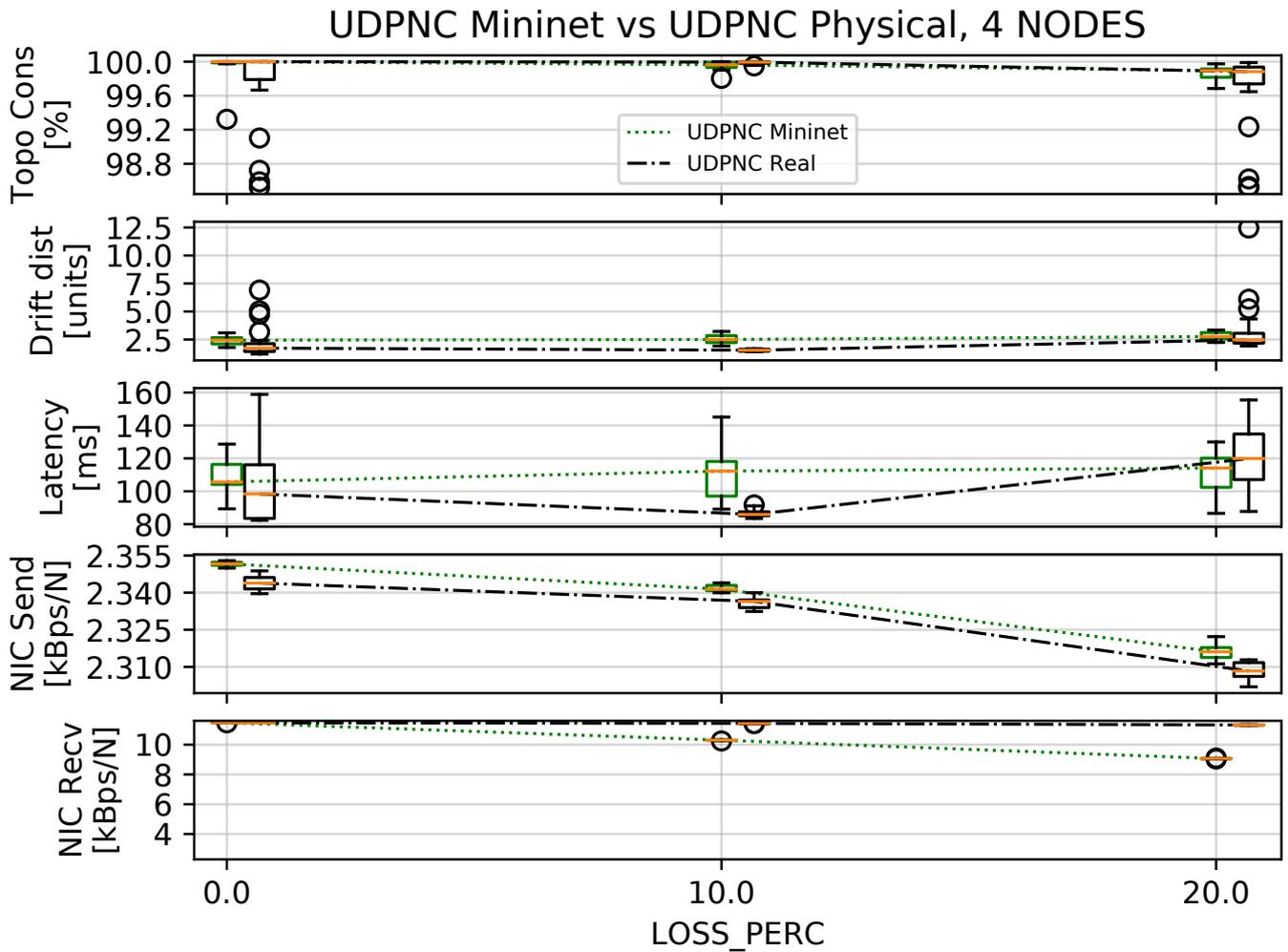


Figure 7.7: Mininet hosts versus physical hosts packet loss test using UDPNC. The Mininet performance is shown in green and the physical network performance is shown in black. Outliers are indicated as black circles.

In accordance with the lower latency, the drift distance performance of UDPNC on physical hosts are also improved.

Finally, the large numbers of outliers in UDPNC performance could be ascribed to packet delay variation (jitter) because of physical wires and hardware. Jitter is not modeled on Mininet networks. Further *hardware* testing could quantify the effect of jitter on the state consistency of a VE and is considered future work.

7.8.4 Summary

UDPNC performance on physical hosts exceed the performance in Mininet due to the distribution of multiple threads to individual CPUs. It is therefore concluded that Mininet performance shows a conservative estimate for UDPNC performance on physical hosts.

7.9 Delay Tests

The delay tests are performed exactly the same as in section 4.13. The purpose of this delay test is to investigate the effect of network delay on UDPNC. We already know from the previous section that UDPNC is a valid packet loss mitigation strategy. However, if UDPNC introduces other adverse network effects, it will have to be reconsidered. All tests were performed at 10% packet loss to represent a wireless environment. Unfortunately Mininet does not model network delay variation. Network delay variation would be possible to measure in a real network scenario and is therefore considered future work.

7.9.1 Experiment Setup

We once again use a custom Mininet topology to specify network delay. Tests are run in the same way as specified in section 7.4.1. Like the packet loss tests above, we only applied the delay after the simulation setup steps (see section 7.2). It should be noted that when we refer to network delay, it is the delay of a packet sent from one network interface to another, without the processing delay. The latency result refers to the latency experienced by VAST messages, which includes the network delay, processing delay and enqueued delay while the packet waits to be processed in the next update step.

In addition to the simulation parameters given in Table 7.1, this specific test uses the simulation parameters given in Table 7.12. A constant packet loss value of 10% was chosen to represent lossy wireless network conditions.

Table 7.12: Delay test simulation parameters

Packet loss	10%
Number of nodes	50
Delay	0 - 200 ms

Table 7.13: Median of average latencies [ms] and 90% confidence interval when adding network delay.

	UDP	UDPNC	\pm %
0.0 ms	130.52 \pm 0.21	111.16 \pm 0.37	-14.83 %
20.0 ms	187.74 \pm 0.50	175.47 \pm 0.42	-6.54 %
50.0 ms	177.54 \pm 0.08	177.16 \pm 0.17	-0.21 %
100.0 ms	298.62 \pm 0.26	294.30 \pm 0.37	-1.45 %
200.0 ms	525.71 \pm 0.44	498.66 \pm 0.34	-5.14 %

7.9.2 Results

From Table 7.13, we can see that UDP and UDPNC latencies are very close (eg. 299 ms versus 294 ms at 100 ms network delay).

From Figure 7.8 it can be seen that UDPNC and UDP perform similarly. In general UDPNC performs slightly better in topology consistency, drift distance and latency.

7.10 Discussion

It is expected (but encouraging) that UDPNC should perform similarly to UDP as the network coding was added to the original UDP network implementation. It is encouraging because these results show that the network coding part of UDPNC does not add extra processing delay while still managing to mitigate the effects of packet loss. The slight improvement in latency can be attributed to the two threaded processing of UDPNC. The improved topology consistency and drift distance can be ascribed to the packet loss mitigation of UDPNC.

7.11 Bandwidth Limited Tests

The bandwidth limit tests were performed exactly the same as in section 4.14. The purpose of this test is to investigate the effect of bandwidth limitations on UDPNC, especially as UDPNC requires more receive bandwidth than UDP or TCP.

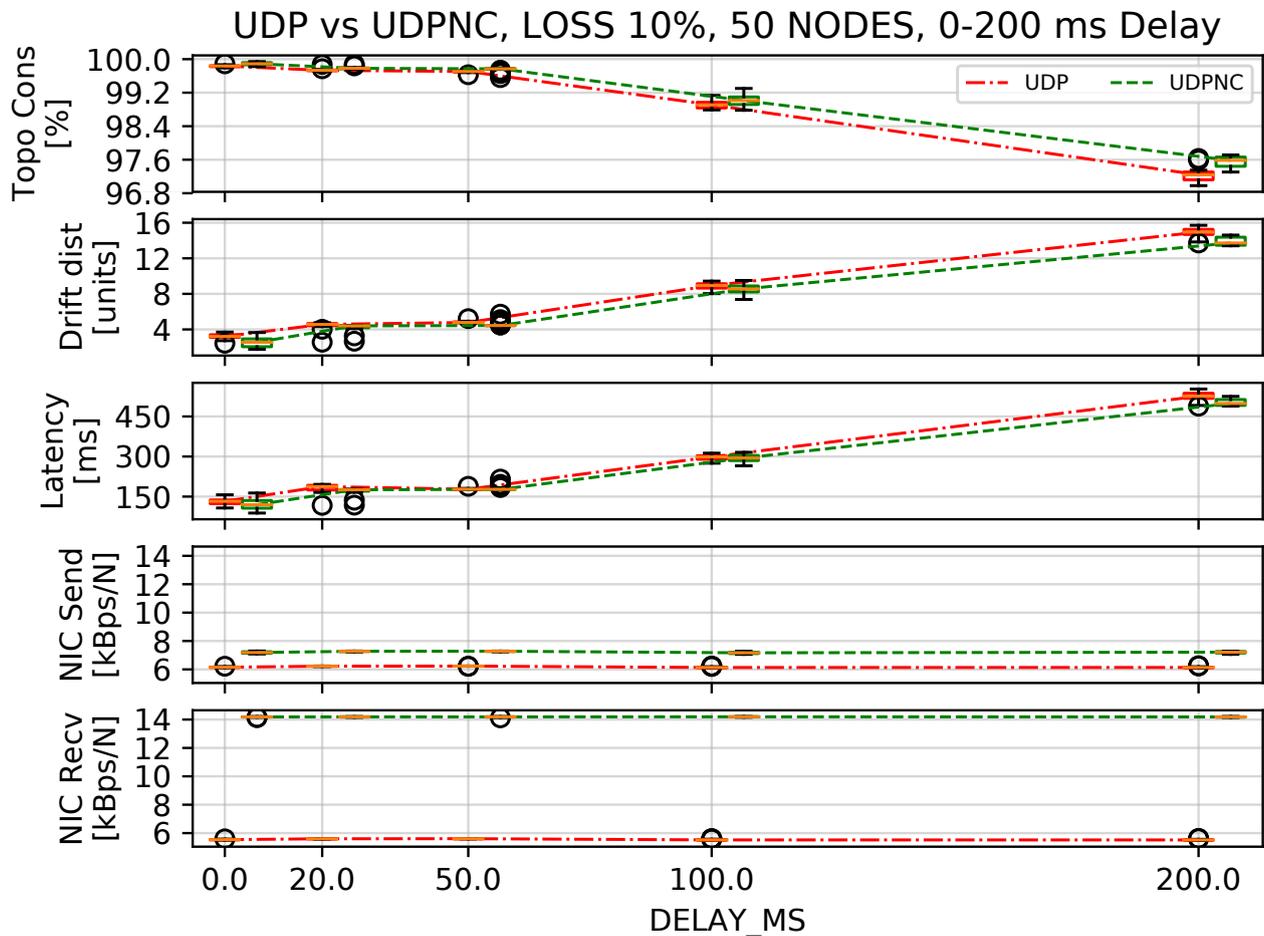


Figure 7.8: Delay tests. Test was run at 10% packet loss with 0 - 200 ms delay. UDP is shown in red and UDPNC in green. Outliers are indicated with black circles.

Table 7.14: Bandwidth limit simulation parameters

Packet loss	0%
Number of nodes	50
Bandwidth limits	1 - 5 MBps

7.11.1 Experiment Setup

The custom Mininet topology in this case specifies the bandwidth limit. Tests are run in the same way as specified in section 7.4.1. We applied the bandwidth limitation after the simulation setup steps.

In addition to the simulation parameters given in Table 7.1, this test uses the simulation parameters given in Table 7.14. Unlike the

Table 7.15: Median of average topology consistencies with 90% confidence interval when limiting bandwidth.

	UDP	UDPNC	\pm %
5.0 MBps	99.99 \pm 0.00	99.98 \pm 0.00	-0.01 %
2.0 MBps	99.99 \pm 0.00	82.40 \pm 0.03	-17.60 %
1.0 MBps	99.99 \pm 0.00	50.85 \pm 0.05	-49.14 %

Table 7.16: Medians of normalized drift distances [VE units] with 90% confidence interval when limiting bandwidth.

	UDP	UDPNC	\pm %
5.0 MBps	2.74 \pm 0.01	3.59 \pm 0.01	30.75 %
2.0 MBps	2.80 \pm 0.01	47.30 \pm 0.02	1590.47 %
1.0 MBps	2.89 \pm 0.01	105.54 \pm 0.08	3551.11 %

Table 7.17: Medians of normalized drift distances [VE units] with degrade percentages when limiting bandwidth.

	UDP	degrade %	UDPNC	degrade %
5.0 MBps	2.74	0.00 %	3.59	0.00 %
2.0 MBps	2.8	2.00 %	47.3	1218.80 %
1.0 MBps	2.89	5.38 %	105.54	2842.59 %

delay test, we decided to not to apply packet loss. This is because the bandwidth limitation has a very sharp decline in performance after 2 MBps for UDPNC and adding packet loss would result in a complete failure of the VAST overlay. Unlike previous bandwidth tests, UDPNC showed such degraded performance that the strictest limitation was set at 1 MBps instead of 0.2 MBps.

The bandwidth limit is applied on each link. The bandwidth requirements in this project is low, as was explained in section 4.4. However, the matcher's links carries the most bandwidth in the network as it needs to receive all of the publications and distribute them to the correct subscribers. Thus even though the bandwidth available in our intended application exceeds the requirement, we perform this test to investigate what would happen in a bandwidth constrained network. UDPNC also requires more bandwidth than UDP or TCP and would therefore be more sensitive to bandwidth constraints.

7.11.2 Results

From Table 7.15 we can see that the performance of UDPNC suffers under bandwidth limitations. Topology consistency is only 50.9%

CHAPTER 7. RESULTS OF USING NETWORK CODING AS PACKET LOSS
168 MITIGATION STRATEGY

Table 7.18: Median of average latencies [ms] and 90% confidence interval when limiting bandwidth.

	UDP	UDPNC	\pm %
5.0 MBps	143.01 \pm 0.52	178.41 \pm 0.35	24.75 %
2.0 MBps	146.52 \pm 0.27	1406.83 \pm 0.27	860.16 %
1.0 MBps	150.03 \pm 0.33	2483.34 \pm 0.63	1555.26 %

Table 7.19: Median of normalised NIC receive bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when limiting bandwidth.

	UDP	total	UDPNC	total	\pm %
5.0 MBps	6.47 \pm 0.00	323.43	15.77 \pm 0.00	788.58	143.81 %
2.0 MBps	6.49 \pm 0.00	324.31	13.56 \pm 0.00	677.95	109.04 %
1.0 MBps	6.49 \pm 0.00	324.7	11.26 \pm 0.00	563.21	73.45 %

at 1 MBps allowed bandwidth. Unlike TCP and UDP which could function with under 1 MBps bandwidth limitations, UDPNC could not communicate under stricter limitations.

From Table 7.16 it can be seen that even at 2 MBps, the drift distance increases rapidly (47 VE units), a degradation of 1219% according to Table 7.17. This also corresponds with the rapid increase in latency from Table 7.18 (1406 ms).

From Table 7.19 we can see that at the bandwidth limit of 2 MBps, the total network bandwidth is 677 kBps, which is still below the allowed bandwidth limit, but the effects of bandwidth limitations are already evident on drift distance and latency performance.

From Figure 7.9 we can see that UDPNC performance severely degrades as bandwidth limitations become stricter.

7.11.3 Discussion

It can be seen that UDPNC is very sensitive to bandwidth limitations. This is due to the coding host generating redundant packets, therefore requiring much more bandwidth than UDP. Bandwidth limitations in constrained networks cause a delay in transmission because packets need to wait in NIC sending queues while the bandwidth is unavailable, leading to increased latency, increased drift distance and lower topology consistency.

As the redundant packets also contend for bandwidth, they exacerbate the congestion. Even if the redundant packets are received successfully, in low packet loss conditions such as these, the packet is redundant and therefore does not bring new state updates.

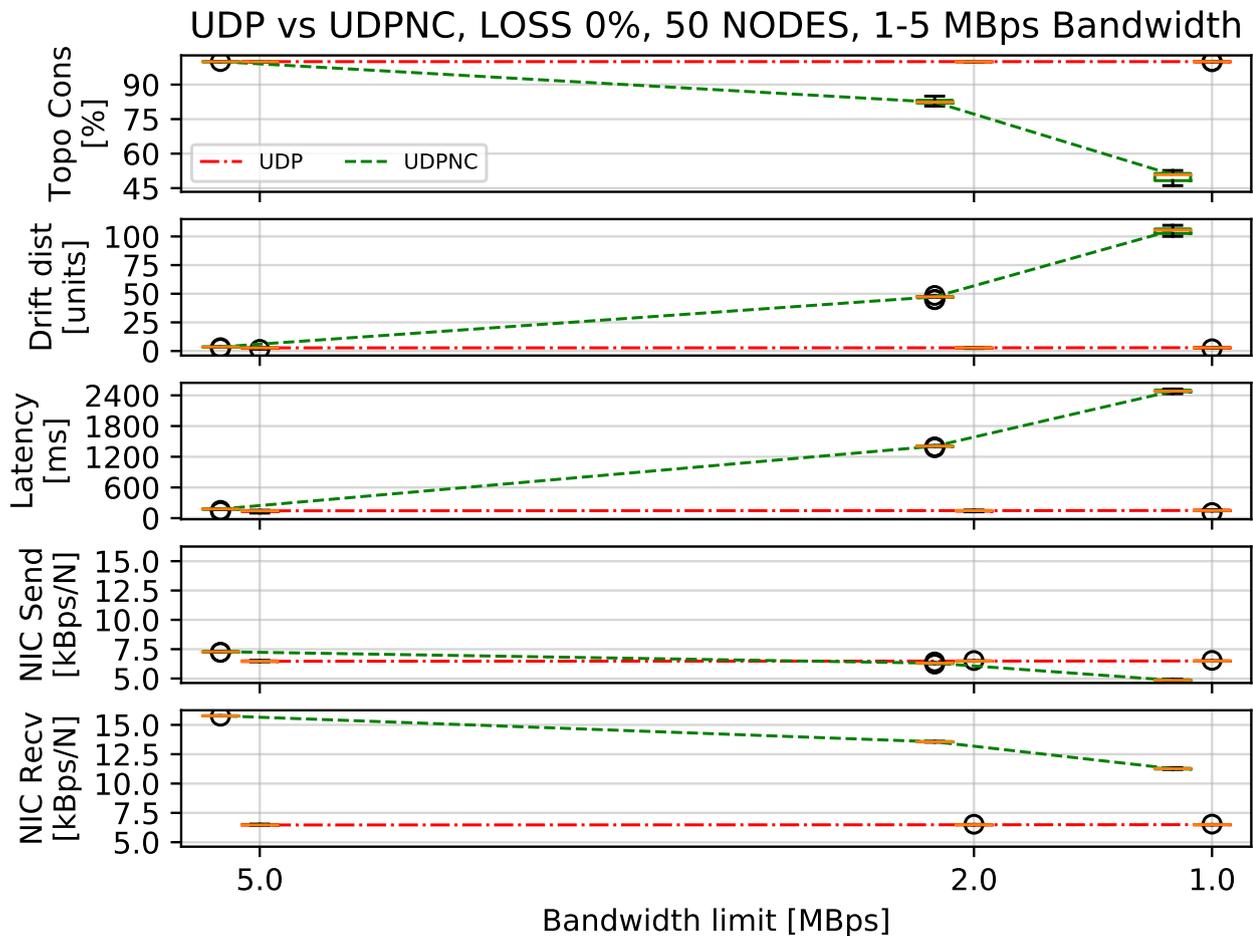


Figure 7.9: Bandwidth limitation tests. UDP results are shown in red and UDPNC in green. The x-axis indicates an increasingly strict limitation on the bandwidth. Outliers are indicated with black circles.

Unexpectedly, the total network bandwidth at the bandwidth limit of 2 MBps is much less than allowed. This can be explained by bursty communication: even though the normalised network bandwidth is below the allowed value, periods of high traffic can come under the bandwidth limitation, delaying those packets. UDPNC is prone to bursty traffic as the coding host generates combined packets whenever they are available, leading to high variations in bandwidth use and therefore frequent limiting.

Table 7.20: Scalability test simulation parameters

Packet loss	10%
Number of nodes	20 - 100
Total Simulation steps:	
20 - 60 nodes	5000
70 - 100 nodes	10000

7.11.4 Summary

UDPNC is very sensitive to bandwidth constraints because of the bursty nature of the coding host traffic as well as the redundant packets requiring extra bandwidth compared to UDP or TCP.

7.12 Scalability Tests

In order to be confident in the usability of UDPNC, we need to evaluate how it performs under scaling conditions. As was mentioned before, we are limited by our simulation setup to 100 nodes running on the same machine.

7.12.1 Experiment setup

In order to represent the typical lossy wireless running conditions, we run the scaling tests at 10% packet loss.

We use a single PC for our Mininet simulations which consisted of 16 CPU cores and 64 GB of RAM. However, during simulations, all of the nodes' processing is done on the same physical machine, which used 90% of the CPU power and 12 GB of RAM with 100 nodes. We therefore limit our test to 100 nodes as larger numbers of processes would likely introduce processing artifacts and distort results.

7.12.2 Results

Table 7.21 shows that topology consistency remains very high when using UDPNC, between 99.89% at 20 nodes and 99.91% at 100 nodes. UDPNC outperforms UDP at more than 20 nodes, especially at 100 nodes: 99.91% versus 99.77%.

Table 7.23 shows that at 20 nodes, UDP latency is lower than UDPNC (100 ms versus 123 ms). However, at 40 nodes and more, UDP latency becomes higher (124 ms versus 119 ms). Furthermore, UDPNC scales very well as, at 100 nodes, it has a average latency of 108 ms. From Figure 7.10 it can also be seen that UDPNC latency remains roughly constant as the number of nodes increase.

Table 7.21: Median of average topology consistencies with 90% confidence interval, for increasing packet loss test.

	UDP	UDPNC	\pm %
20	99.90 \pm 0.00	99.89 \pm 0.00	-0.01 %
30	99.89 \pm 0.00	99.90 \pm 0.00	0.01 %
40	99.86 \pm 0.00	99.90 \pm 0.00	0.04 %
50	99.84 \pm 0.00	99.89 \pm 0.00	0.05 %
60	99.84 \pm 0.00	99.92 \pm 0.00	0.08 %
70	99.80 \pm 0.00	99.92 \pm 0.00	0.11 %
80	99.80 \pm 0.00	99.92 \pm 0.00	0.13 %
90	99.79 \pm 0.00	99.92 \pm 0.00	0.13 %
100	99.77 \pm 0.00	99.91 \pm 0.00	0.14 %

Table 7.22: Medians of normalized drift distances [VE units] with 90% confidence interval when scaling number of nodes.

# Nodes	UDP	UDPNC	\pm %
20	2.45 \pm 0.00	2.81 \pm 0.01	14.55 %
30	2.64 \pm 0.01	2.54 \pm 0.01	-3.68 %
40	2.95 \pm 0.01	2.44 \pm 0.01	-17.16 %
50	3.21 \pm 0.00	2.55 \pm 0.01	-20.65 %
60	3.34 \pm 0.01	2.17 \pm 0.01	-34.96 %
70	3.59 \pm 0.01	2.13 \pm 0.01	-40.67 %
80	3.78 \pm 0.01	2.01 \pm 0.00	-46.96 %
90	3.75 \pm 0.01	1.96 \pm 0.01	-47.80 %
100	3.94 \pm 0.01	2.06 \pm 0.00	-47.69 %

Table 7.23: Median of average latencies [ms] and 90% confidence interval when scaling number of nodes.

# Nodes	UDP	UDPNC	\pm %
20	100.19 \pm 0.07	123.43 \pm 0.26	23.20 %
30	109.67 \pm 0.21	121.44 \pm 0.24	10.73 %
40	124.42 \pm 0.31	119.51 \pm 0.31	-3.95 %
50	130.52 \pm 0.21	111.16 \pm 0.37	-14.83 %
60	143.93 \pm 0.40	114.92 \pm 0.34	-20.16 %
70	151.80 \pm 0.22	110.98 \pm 0.27	-26.89 %
80	159.09 \pm 0.22	106.91 \pm 0.16	-32.80 %
90	158.80 \pm 0.27	105.75 \pm 0.21	-33.40 %
100	165.73 \pm 0.26	108.20 \pm 0.20	-34.71 %

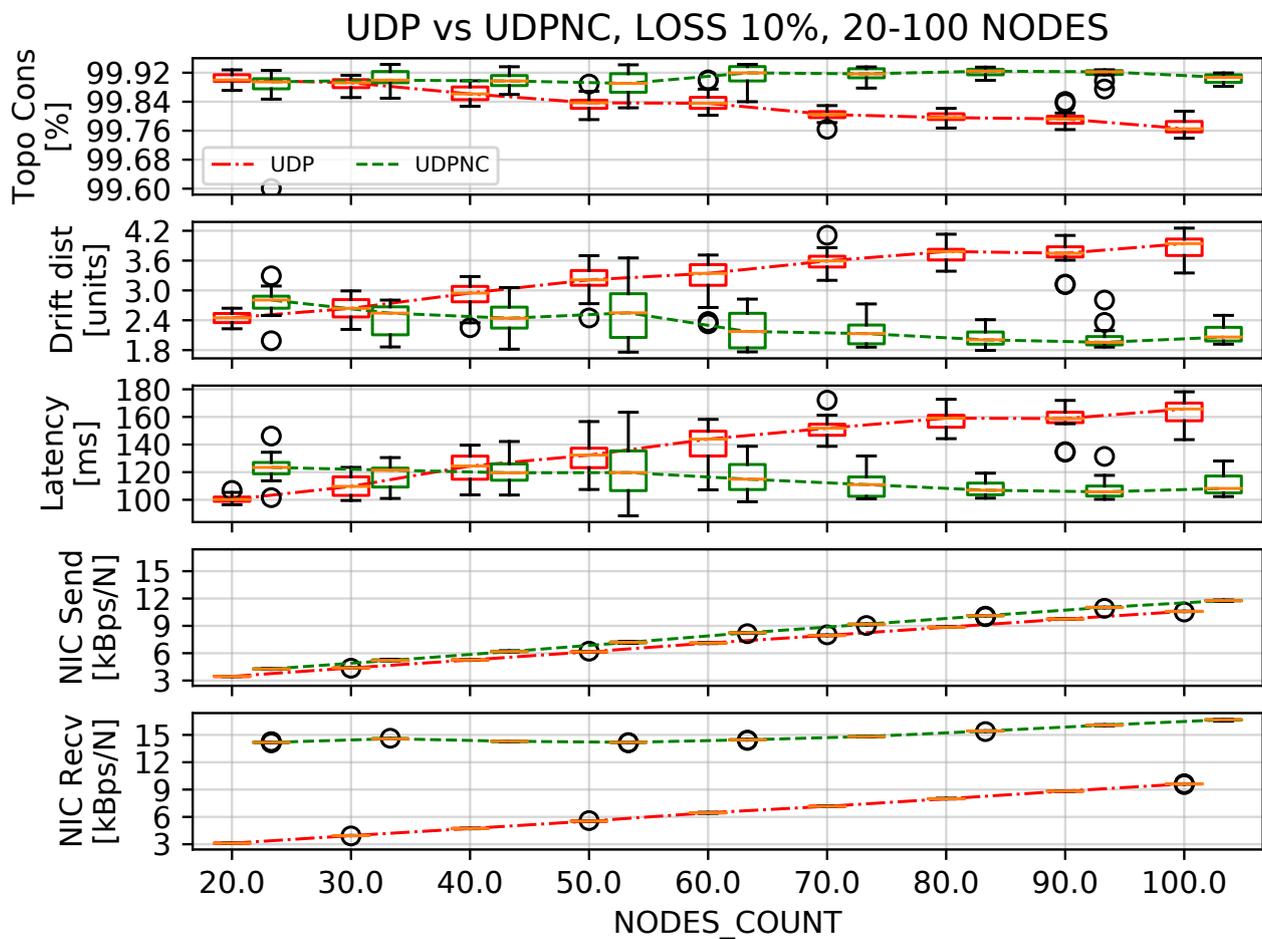


Figure 7.10: Scaling test with 20 - 100 nodes. UDP results are shown in red and UDPNC results in green. All tests were run at a packet loss percentage of 10% and zero delay. Outliers are indicated with black circles.

Table 7.24: Median of normalised NIC send bandwidth [kBps/node], 90% confidence interval and total bandwidth of the network when scaling number of nodes.

# Nodes	UDP	total	UDPNC	total	\pm %
20	3.45 \pm 0.00	69.1	4.27 \pm 0.00	85.46	23.67 %
30	4.38 \pm 0.00	131.46	5.20 \pm 0.00	156.1	18.75 %
40	5.25 \pm 0.00	210.19	6.18 \pm 0.00	247.17	17.59 %
50	6.15 \pm 0.00	307.63	7.19 \pm 0.00	359.61	16.90 %
60	7.11 \pm 0.00	426.57	8.24 \pm 0.00	494.43	15.91 %
70	7.95 \pm 0.00	556.55	9.18 \pm 0.00	642.41	15.43 %
80	8.85 \pm 0.00	707.99	10.12 \pm 0.00	809.69	14.36 %
90	9.75 \pm 0.00	877.35	11.02 \pm 0.00	992.09	13.08 %
100	10.59 \pm 0.00	1059.05	11.77 \pm 0.00	1177.46	11.18 %

Table 7.22 shows similar results for drift distance: at 20 nodes UDP drift distance is lower (2.45 versus 2.81 VE units) but at 40 nodes UDP drift distance is higher (2.95 versus 2.44 VE units). UDPNC scales well with the drift distance at 100 nodes of 2.06 VE units.

Table 7.24 shows NIC send bandwidth increases as more nodes are present in the network, according to expectation.

7.12.3 Discussion

UDPNC performs similarly to UDP even with more nodes in the VE. At higher node numbers, UDPNC outperforms UDP, mostly due to the lower latency.

The topology, drift distance and latency results are surprising: at a low number of nodes, UDP has a lower latency when compared to UDPNC. This can be explained by two effects: firstly, the two-threaded nature of UDPNC results in processing multiple incoming queued packets quicker than UDP. As was mentioned before, note that latency is measured by the VAST application and therefore includes queuing and processing times.

Secondly, and in contrast to the first effect, UDPNC has a slightly longer processing time *per packet* compared to UDP. Therefore, it can be concluded that multi-threaded UDPNC only starts showing benefits at many nodes, where multiple packets are waiting in the queue. At lower node numbers the processing delay dominates the latency results.

Furthermore, UDPNC scales very well, with better topology consistency, drift distance and latency results at *more* nodes in the network. We can explain this by the interactions on the coding host:

packets from two different hosts are required in order to generate one encoded packet. In a sparse VE (for example only 20 nodes), the arrival frequency of packets at the coding host is low, and therefore the packets need to wait longer in order to be encoded. In contrast, with 100 nodes, many packets arrive at the coding host, possibly filling up its packet pool. There is thus more frequently packets to encode, allowing redundancy packets to be dispatched sooner. The packets therefore experience a lower latency with higher numbers of nodes.

7.12.4 Summary

In terms of latency, UDPNC scales better than UDP because of the two-threaded packet processing and higher packet availability on the coding host. However, larger receive bandwidth requirements could limit the scalability of UDPNC in bandwidth constrained environments.

7.13 Cost of UDPNC

The most significant result above is the increased receive bandwidth of UDPNC. The UDPNC network implementation will always receive more packets as redundancy is the mechanism by which the effect of packet loss is reduced. However, this could also limit the scaling opportunities of the UDPNC implementation as it will saturate the available network bandwidth much quicker than the other implementations. This would be a trade-off that needs to be considered in a real world system. Dynamic scaling of redundancy depending on network conditions could be used to alleviate network saturation, but UDPNC will always require more bandwidth in order to provide packet loss mitigation.

7.14 Emulated Network Model Inaccuracies and Consequences

In this section we discuss the inaccuracies that using an emulated network introduced in our simulation.

7.14.1 Shared Wireless Channel Contention

Our Mininet emulated network does not model the contention mechanisms of a shared wireless channel and therefore our latency mea-

surements could be not completely representative of tests run on a real wireless network. Ideally, real hardware should be used to determine the effect of 802.11 MAC interactions. Despite this, the emulated packet loss in Mininet is still enough to provide us with valuable insights into the effect of packet loss and possible mitigation strategies to use in real wireless networks.

On the other hand, the shared wireless channel can also be used to the advantage of UDPNC: by broadcasting the redundant coded packets, recovery packets can be disseminated efficiently.

Thus the shared wireless channel provides challenges and opportunities for UDPNC.

7.14.2 Wireless Retransmission

In a typical Wi-Fi 802.11n network, retransmissions are used. Retransmissions both reduce the packet loss experienced by the transport layer as well as possibly creating more transmission delay. Our emulated network does not perform retransmissions and emulate a high loss rate (up to 70%). This is comparable to running Wi-Fi with CRC checks disabled [110], but this would require changes to MAC layer configuration in a real Wi-Fi network.

7.15 Benefits of Mininet Emulated Network

However, despite the above mentioned inaccuracies, the following advantages were obtained by using an emulated network:

- Accurate control over the network conditions such as packet loss, delay and bandwidth limitations. This allows for the experiments to be completed with high accuracy and repeatability.
- Timing is exactly synchronised, as all of the local processes uses the same system clock.
- Using Mininet and an OpenFlow controller (POX), our software can run on a real OpenFlow network without alterations.

7.16 Summary

In this chapter we presented the results of UDPNC tested in adverse network conditions. We compared these results with the UDP imple-

*CHAPTER 7. RESULTS OF USING NETWORK CODING AS PACKET LOSS
176 MITIGATION STRATEGY*

mentation which previously performed the best under adverse conditions. We showed that UDPNC could improve reliability at the cost of extra bandwidth. We also showed that other adverse network conditions, except notably bandwidth limitations, do not comparatively affect UDPNC more than UDP. That is, UDPNC performs as well as UDP without any extra considerations except increased bandwidth. Finally we discussed the short-comings of our emulated network and how our results could differ from a real wireless network.

Chapter 8

Conclusions and Future Work

In the introduction we presented the problem statement:

How can the state consistency of a Virtual Environment be maintained under adverse network conditions?

To answer this we considered two research questions:

- How does adverse network conditions affect state consistency of a virtual environment?
- Is network coding an effective packet loss mitigation strategy?

which we will now answer based on the results presented in the previous chapters.

8.1 How Does Adverse Network Conditions Affect The State Consistency of a Virtual Environment?

From the TCP and UDP tests in Chapter 4 we could see that almost any adverse network condition affects state consistency. The drift distances and degradation percentages observed are summarised in Table 8.1.

Table 8.1: Summary of drift distance and degradation percentages under adverse conditions.

Adverse condition	TCP	degrade %	UDP	degrade %
70% packet loss	246.36	11388.67 %	55.12	1924.35 %
200 ms network delay	41.94	573.75 %	14.95	365.26 %
0.2 MBps bandwidth	257.8	9841.11 %	121.58	4332.29 %

It is not possible to directly compare the degradation percentages as the network conditions affect the state consistency differently. However, it can be concluded that adverse network conditions degrade state consistency, with UDP degradation typically less than TCP.

Although not an adverse network condition (and therefore not considered degradation) increasing the number of nodes in the network does influence the state consistency: UDP saw the most significant increase in drift distance (3.94 VE units at 100 nodes, 60.8% increase). TCP drift distance remains roughly the same (5.80 VE units at 100 nodes versus 5.84 VE units at 20 nodes). However, neither increase was on the scale of the other adverse network conditions, possibly because we were limited to simulating only 100 nodes.

From our evaluation of TCP as a transport protocol we concluded that adverse conditions severely degrades the performance of TCP, possibly due to the retransmission and flow control unoptimised for such conditions. Although retransmissions are expected to improve reliability, in practice, it reduces reliability even further under *adverse* conditions, i.e. high packet loss, long delays or strict bandwidth limitations.

Although interactive applications typically already use UDP protocols for the transport layer, it is useful to know that UDP is also the best suited for interactive applications in *adverse* network conditions.

8.1.1 Summary

To summarise, traditional retransmission reliability mechanisms such as TCP are not suitable for adverse network conditions. Adverse network conditions affect the state consistency and could potentially degrade the user experience.

8.2 Is Network Coding an Effective Packet Loss Mitigation Strategy?

From the UDP and UDPNC tests in Chapter 7 we could see that UDPNC can provide improved reliability in packet loss conditions while not degrading performance under other network conditions, except bandwidth limitations.

The most significant improvements of UDPNC over UDP is shown in Table 8.2: at 30% packet loss, 29.3% lower drift distance and at 70% packet loss, topology consistency is 9.09% higher.

8.2. IS NETWORK CODING AN EFFECTIVE PACKET LOSS MITIGATION STRATEGY?

179

Table 8.2: Summary of best topology consistency and drift distance UDPNC mitigation under packet loss.

	Topology consistency			Drift distance		
	UDP	UDPNC	\pm %	UDP	UDPNC	\pm %
30.0 %	98.80	99.27	0.48 %	5.78	4.09	-29.30 %
70.0 %	71.04	77.50	9.09 %	55.12	54.00	-2.04 %

In the delay and scalability tests UDPNC performed as least as well as UDP, typically slightly better: at 200 ms network delay, UDPNC latency was 5.14% lower and at 100 nodes, 47.7% lower. This shows that the network coding part of UDPNC does not introduce excessive processing delay and that UDPNC does not degrade more than UDP under other adverse network conditions.

The disadvantage of UDPNC is the bandwidth requirements: although UDPNC only sends around 12% more bytes than UDP, the receiving bandwidth of UDPNC is 143% higher than that of UDP. This is due to the extra packets that are generated for redundancy. Under normal network conditions this is acceptable with a small number of nodes in the network. However it could become problematic with many (100+) nodes in the VE.

It is especially problematic when there is a bandwidth restriction on the network. For example at a bandwidth limitation of 1 MBps, UDP drift distance performance was still close to nominal (2.89 VE units with a degradation of 5.38 %), but UDPNC performance degraded severely (105.54 VE units with a degradation of 2842.59 %).

UDPNC is an effective packet loss mitigation strategy but at the cost of much higher bandwidth. Using the state consistency metrics we can show that UDPNC better maintains the VE state consistency under adverse network conditions, compared to other networking implementations.

8.2.1 Summary

In our evaluations, we have shown that UDPNC provides increased state consistency under packet loss conditions. Network coding based reliability mechanisms should strongly be considered when implementing an interactive application such as a VE or game under adverse conditions (for example wireless networks).

8.3 VAST Case Study and Limitations

In this dissertation we used VAST as a case study for investigating the effects of adverse network conditions on state consistency and testing the potential of network coding for mitigating packet loss. We propose that our results are generalisable to the state consistency of movement updates in similar VE applications which are typically sent in small packets. However, generalisation is limited in the following ways:

- VAST only handles position updates as state. In real VEs, many other variables are included in state, eg. user and entity characteristics or dynamic environment layout. Any property of the VE which can be altered should be included in the state. For example, if a door is opened by a player, the next player to arrive at that position in the VE should see a closed door, and should be able to open it for the next player to pass by. Adverse network effects will influence the state consistency of movement updates and environment updates differently as changes to environment could occur less often than movement updates. If such a low frequency update would be lost, the state could remain inconsistent for a long time.
- Currently VAST only implements random walk movements or random clustering movements. In this dissertation we only considered random walk movement. According to Liang et al. [83], typical movement patterns are more complex than randomised movement and vary depending on the type of VE. For example, First-Person Shooter game players are constantly moving in the accessible regions, while MMORPG players typically move towards objectives (clustering around interest points). Furthermore, the movements are not uniform, that is players move around the VE at different speeds depending on their immediate surroundings. All of these properties will influence the network traffic generation. For example, more players will interact at points of interest, causing higher bandwidth use to the servers or peers servicing those regions. Non-uniform movement could cause bursty traffic.
- VAST uses event-based messaging in order to achieve state consistency. This has two drawbacks: the number of messages for clients in close proximity could grow with $\mathcal{O}(N^2)$ and there is no persistence of state if a node should leave. Even though VAST uses SPS to reduce the number of event messages to those clients interested, if the clients cluster together, very little

Table 8.3: Summary of related work on VAST

Property	Previous result	Our result	Comments	From
Average bandwidth	2.5 kBps	5.07 kBps	Twice the AOI size was used.	[69]
Average latency	120 ms	118.8 ms		[71]
Drift distance with 100 nodes	2	2.39		[75]
Average bandwidth with 100 nodes	6kBps	9.69 kBps	AOI and node number similar, unknown client movement speed	[44]

benefit is achieved from SPS, because all messages still need to be dispatched by the matcher. Hu suggests that state managers or game logic servers can be included in the network in order to implement update-based consistency (reducing the number of messages that need to be sent) and also to provide storage for state persistency. In this project the traffic associated with state managers or game logic servers were not included. Furthermore, this project uses only one Matcher and therefore the inter-matcher traffic is also not included.

All of the above mentioned limitations decrease the generalisability of the results found in this project. However, we propose that valuable insights were still obtained and the limitations create opportunities for improvements and further testing in future work.

8.4 Comparison to Prior Work

8.4.1 Performance Comparison with Related Work in State Consistency

Hu et al's papers [69], [71] as well as papers by Jiang, Huang and Hu [75] and Chen, Lin, Chen and Hu [44] are the closest comparisons in literature. As was described in section 4.11.8 and sections 4.15.4 and summarised in Table 8.3, our simulations perform similar to previous work on VAST.

From Table 8.3 it can be seen that our Mininet simulations match results from previous work. This implies that Mininet network emulation is a useful and accurate testbed for experimentation of VE

Table 8.4: Summary of experience latency under adverse network conditions compared to related work.

Condition	From [47]		Our results		
	MQTT	CoAP	TCP	UDP	UDPNC
Network latency 200 ms	750 ms	200 ms	878 ms	525 ms	498 ms
10% Packet loss	200 ms	≈ 0 ms	147 ms	132 ms	119 ms

state consistency methodologies. As implementations and topologies that run on Mininet can also be transferred to real networks, Mininet is a valuable middle ground between pure simulation and real networks.

8.4.2 Performance Comparison with Related Work in Adverse Network Conditions

In this section we will compare our performance under adverse conditions to related work. Note that absolute comparisons are not possible, as the research domain, network setup and applications influence performance.

8.4.2.1 Constrained Network Conditions on IoT Protocols

In [47], Chen and Kunz investigate the effect of adverse network conditions, like packet loss, network delay and bandwidth limitations on the latency experienced by IoT protocols such as MQ Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP).

MQTT relies on TCP as its transport protocol and CoAP on UDP. Table 8.4 shows the performance related to our results. Results show that MQTT suffers similar degradation as TCP did in our VE simulation. CoAP experiences less degradation than MQTT, similar to our UDP results. It is important to note that both MQTT and CoAP transport small data packets used for IoT, at an average data rate of 1.64 kbps, much smaller than the total amount of data on our network.

From this paper we can see that adverse network conditions have similar effects on transmission, despite the domain difference. This gives rise to the possibility that our packet loss mitigation strategy could also be applied in areas other than virtual environments.

8.4.2.2 Latency in First-Person Shooter Games

Beigbeder et al. [34] present a study on the effects of packet loss and network delay on user performance in a First-Person Shooter (FPS)

Table 8.5: Summary of network delay results.

	TCP	UDP
100.0 ms	188.84	298.62
200.0 ms	878.28	525.71

game. In a pilot study, they evaluated the packet loss rates and network delay to network FPS servers during operation. They found that connections to most servers did not have packet loss and the maximum packet loss was 3%. In terms of network delay they found that 40% of servers had below 100 ms, 40% 100 - 140 ms and the remaining 20% had more than 140 ms network delay. From these results, they decided to test user performance under 0-6% packet loss and 0 - 400 ms network delay.

In the user performance test they found that FPS hit fractions degraded by 40% under 200 ms network delay. They could not determine a relationship between packet loss and hit fraction, either because the packet loss did not influence the user experience greatly, or a dead-reckoning mechanism is already built into the FPS game to reduce the effect of packet loss.

From our latency test results, summarised in Table 8.5, we can see that 100 ms network delay would already induce an experienced latency of 188 and 298 ms latency for TCP and UDP respectively. We predict that at 200 ms network delay, the experienced latency would be so high as to make the VE almost unusable.

Beigbender also concludes that 100 ms of network delay would be noticeable and 200 ms would be disruptive.

8.4.2.3 Latency in VR

According to Elbamby, Perfecto, Bennis and Doppler [53], the requirements for latency is less than 20 ms to avoid motion sickness. Under nominal network conditions, our results were 115, 142 and 124 ms for TCP, UDP and UDPNC respectively. This would have been unacceptable if our system needed to stream VR content directly to a Head Mounted Device. However, these latencies include the waiting time for the next step, before packets are processed. Stepwise event generation and processing is typical for MMVE systems, but should be adapted for immersive VR. For example, if the step duration in our simulation was decreased to 10 ms or even 1 ms, experienced latencies could be reduced.

Furthermore, if dead-reckoning is used, the frame rate of the VR device could be much higher without the user experiencing the state

inconsistency while state updates could arrive at a slower pace if the network cannot achieve the required bandwidth.

Considering the relative latencies in Table 8.4, we can calculate that UDPNC has 28 ms and 13 ms less latency under 10% packet loss conditions than TCP and UDP respectively. Therefore it can be surmised that the relative reduction of latency by using UDPNC can improve performance enough to reduce motion sickness.

From this paper we can see that network delay still needs to be reduced significantly in order to meet the required latencies of VR. Furthermore, VE implementations will have to be specialised in order to provide state consistency at much higher update frequencies.

8.5 Concluding Remarks

We argue that both our research questions were addressed by quantifying how adverse network conditions degrades state consistency as well as proving that network coding can assist in recovery under packet loss conditions. Network Coding and UDP perform well together and could be the basis for a semi-reliable, time-sensitive and distributed protocol in the future.

Furthermore, we show that our work agrees with prior work using the VAST library by Hu and others. From the papers on network conditions we see that other domains experience similar degradation as our evaluations showed. Finally, from the paper on latency in VR, it is clear that more research into state consistency management at high update frequencies and improved network delay will be needed in order to meet the requirements of fully immersive, wireless VR.

8.6 Future Work

In this section we will expand on how UDPNC can be improved and further realistic testing can be done to further quantify the effect of adverse network conditions. We then look at other ways of using UDPNC, i.e. to improve throughput and how UDPNC can be implemented outside of the VAST library.

8.6.1 UDPNC Improvements

The biggest improvement to UDPNC would be to reduce the protocol overhead. Currently encoded packet size is fixed at 500 bytes to accommodate most VAST message sizes. However if the encoded packet can be sized exactly, i.e. the smaller packets are zero padded

only to the length of the largest packet to be encoded, sending encoded packets would be more bandwidth efficient and increase redundancy efficiency. Only small changes to existing code would be needed: by setting the packet size field to the largest packet size, a variable size packet can be defined. The RLNC decoder can then use this field to determine the correct symbol size for decoding.

8.6.2 Adaptive Network Coding

In addition to variable encoded packet sizes, adaptive redundant packet generation could be useful as well. In high packet loss conditions, a single redundant packet for every two packets was observed to be insufficient to mitigate the loss. In such cases it could be useful to generate more redundant packets, possibly in multiple rounds of redundant transmissions and using wireless overhearing. This would require changes to both the coding host and UDPNC implementation and requiring that packets are kept for longer in the packet pool (explained in section 6.5.2.1).

8.6.3 Security and Privacy

Security and privacy in peer-to-peer networks was not considered in this dissertation. However, such considerations would be needed if the VE application includes many users, some of whom might try to cheat or disrupt the VE application for their own gains. We would refer the reader to the survey paper by Buyukkaya et al [39] as well as work in [42] and [85].

8.6.4 Larger, Realistic Hardware Test

In order to verify the results of the TCP, UDP and UDPNC in general, a larger test would be required, preferably on a real network. One of the limitations of this project was that we did not have access to a OpenFlow enabled switch, but rather emulated the network using Mininet. We also used a PC with four network ports in order to establish a small network with four physical hosts and a coding host. The four host network was a proof-of-concept, showing that the software could run on any OpenFlow switch. However, the scaling effects of the Virtual Environment could not be tested with only four hosts.

Ideally a full Wi-Fi stack should be used, where redundancy would also be sent as a single transmission over the air (using wireless overhearing), instead of multiple unicasts. This would truly demonstrate the gains of using network coding in wireless scenarios.

Future tests should also include more matchers in the VAST overlay. Due to packet loss problems, only one matcher was used in our simulations. However, using higher reliability for inter-matcher communication, the scaling opportunities of super-peer-to-peer operation can once again be leveraged.

Finally, the effect of network delay variation (jitter) on state consistency should be measured.

8.6.5 Further Protocol Comparisons

The available protocols are ever increasing and comparing protocols could uncover benefits or applications previously overlooked. For example, the Stream Control Transmission Protocol (SCTP) and the QUIC (HTTP over UDP) protocols could possibly be used as a transport protocol in VE applications. In the same way the TCP improvements recommended by [64] and [98] should be implemented and compared with results from this dissertation.

8.6.6 Realistic Mobility Models

Liang [83] and Carlini [40] present studies of typical movement patterns in VEs and possible implementations that can be used in VE simulations such as ours. In order to generate network traffic representative of actual user movement, such tools will need to be incorporated in the simulation. We expect that actual user movements could change average bandwidth as well as generate more bursty traffic patterns, which could, at worst, reduce the network coding opportunities or cause congestion.

8.6.7 Bandwidth Reduction using Interflow Network Coding

Network Coding can also be used to improve throughput or conversely reduce bandwidth. In low packet loss environments interflow network coding could be used to reduce bandwidth and increase scalability while in higher loss environments network coding can be switched to reliability improvement. It could even be possible to dynamically generate interflow and reliability packets at the same time, depending on current network conditions.

8.6.8 Beyond VAST

Currently UDPNC is implemented as a part of the VAST library. However it is mostly application agnostic and should be applicable to

other Virtual Environment applications as well as other interactive applications.

Ideally UDPNC should be implemented as a middle-ware in order to provide reliability for a number of packets. This could be accomplished by either capturing outgoing packets with a software such as libpcap [86] or by forwarding packets through a network tunnel device administrated by the UDPNC middle-ware.

8.7 Future Research Questions

In this section we will discuss possible future research questions that this work has uncovered.

8.7.1 State Consistency under Adverse 802.11 Wi-Fi Communication

The premise of this dissertation is that Wi-Fi communication is inherently lossy and could also be subject to network delay. Wi-Fi, especially 802.11ac is not bandwidth constrained (typically 600 - 850 Mbps [112] bandwidth can be achieved), and therefore packet loss mitigation strategies like UDPNC could be applicable.

If VR is to become fully immersive as well as wireless, with Wi-Fi possibly the data link layer, we need to determine the performance of interactive applications over Wi-Fi. Mininet does not allow us to measure the effect of network delay variation or Wi-Fi retransmissions on state consistency. Network delay variations and retransmissions are highly likely in high usage Wi-Fi contention scenarios, which could degrade the state consistency. We phrase the question as follows:

What is the state consistency of a VE application under 802.11ac Wi-Fi communication, with and without UDPNC as transport layer and how does UDPNC respond to network delay variations?

8.7.2 Reliable Interactive Protocol

UDPNC still needs many improvements to be become the defacto reliability transport protocol. Ideally, a reliable protocol with performance comparable to TCP is needed for adverse network conditions. Practically, a protocol cannot be truly reliable under adverse conditions. However, if the theoretical transport layer protocol could be sensitive to the Quality-of-Service (QoS) required of the data to

transport, signaled by the application, network coded redundancy could be adaptively added on a message-by-message basis. Also, reliability could be exchanged for latency in cases where messages are not as latency sensitive. However, adding network coded redundancy dynamically could possibly require the co-ordination with intermediate nodes. Such a protocol could also be network sensitive and communicate the QoS level available back to the application. We propose the following research question:

What reliability mechanisms / redundancy rates would be required to guarantee an average Quality-of-Service in a given adverse network condition? What are the trade-offs between reliability and latency? And what coordination would that require from intermediate nodes / routers in the network?

Bibliography

- [1] “ACE Overview.” [Online]. Available: <http://www.dre.vanderbilt.edu/~schmidt/ACE-overview.html>
- [2] “Boost C++ Libraries.” [Online]. Available: <https://www.boost.org/>
- [3] “CUBIC for Fast Long-Distance Networks - RF8312.” [Online]. Available: <https://tools.ietf.org/html/rfc8312>
- [4] “Facebook announces Horizon, a VR massive-multiplayer world.” [Online]. Available: <https://techcrunch.com/2019/09/25/facebook-horizon/>
- [5] “Introduction to Mininet.” [Online]. Available: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>
- [6] “iptables(8) - Linux man page.” [Online]. Available: <https://linux.die.net/man/8/iptables>
- [7] “Linux Man pages: TC NetEm - Network Emulator.” [Online]. Available: <http://man7.org/linux/man-pages/man8/tc-netem.8.html>
- [8] “Min-cut Max-flow.” [Online]. Available: https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem
- [9] “Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet.” [Online]. Available: <http://mininet.org/>
- [10] “Ns-2 Network Simulator.” [Online]. Available: http://nsgn.sourceforge.net/wiki/index.php/User_Information
- [11] “Oculus Quest review.” [Online]. Available: <https://www.techradar.com/reviews/oculus-quest-review>
- [12] “Open vSwitch.” [Online]. Available: <https://www.openvswitch.org/>

- [13] “OpenFlow - Open Networking Foundation.” [Online]. Available: <https://www.opennetworking.org/sdn-resources/openflow>
- [14] “RFC 4318 - Definitions of Managed Objects for Bridges with Rapid Spanning Tree Protocol.” [Online]. Available: <https://tools.ietf.org/html/rfc4318>
- [15] “RFC 5905 - Network Time Protocol Version 4: Protocol and Algorithms Specification.” [Online]. Available: <https://tools.ietf.org/html/rfc5905>
- [16] “RFC 6330 - RaptorQ Forward Error Correction Scheme for Object Delivery.” [Online]. Available: <https://tools.ietf.org/html/rfc6330>
- [17] “RFC 826 - An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware.” [Online]. Available: <https://tools.ietf.org/html/rfc826>
- [18] “Stream Control Transmission Protocol RFC4960.” [Online]. Available: <https://tools.ietf.org/html/rfc4960>
- [19] “The Best VR Headsets for 2019.” [Online]. Available: <https://www.pcmag.com/article/342537/the-best-virtual-reality-vr-headsets>
- [20] “The Inside Story of Oculus Rift and How Virtual Reality Became Reality.” [Online]. Available: <https://www.wired.com/2014/05/oculus-rift-4/>
- [21] “The POX network software platform.” [Online]. Available: <https://github.com/noxrepo/pox/>
- [22] “Top 6 Most Popular MMORPGs Sorted by Population (2019) | Altar of Gaming.” [Online]. Available: <https://altarofgaming.com/all-mmos-sorted-by-population-2018/>
- [23] “VAST - scalable peer-to-peer (P2P) network for virtual environments (virtual worlds, MMOG and simulations).” [Online]. Available: <http://vast.sourceforge.net/>
- [24] “What About Stream Control Transmission Protocol (SCTP)? | Network World.” [Online]. Available: <https://www.networkworld.com/article/2222277/what-about-stream-control-transmission-protocol--sctp--.html>

- [25] “Windows Network Architecture and the OSI Model - Windows drivers | Microsoft Docs.” [Online]. Available: <https://docs.microsoft.com/en-gb/windows-hardware/drivers/network/windows-network-architecture-and-the-osi-model>
- [26] “Wireshark · Go Deep.” [Online]. Available: <https://www.wireshark.org/>
- [27] “The Games With The Biggest Player Base,” 2019. [Online]. Available: <https://esportsjunkie.com/2019/08/05/the-games-with-the-biggest-player-base/>
- [28] O. Abari, D. Bharadia, A. Duffield, and D. Katabi, “Cutting the Cord in Virtual Reality,” *Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets '16*, pp. 162–168, 2016. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=3005745.3005770>
- [29] R. Abdelmoumen, M. Malli, and C. Barakat, “Analysis of TCP latency over wireless links supporting FEC/ARQ-SR for error recovery,” *IEEE International Conference on Communications*, vol. 7, no. c, pp. 3994–3998, 2004.
- [30] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, “Network information flow,” *{IEEE} Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
- [31] J. Arkkio, “QUIC transport protocol: designed for today’s bandwidth-hungry applications - Ericsson.” [Online]. Available: <https://www.ericsson.com/en/blog/2019/1/quic>
- [32] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. [Online]. Available: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>
- [33] E. Bastug, M. Bennis, M. Medard, and M. Debbah, “Toward Interconnected Virtual Reality: Opportunities, Challenges, and Enablers,” *IEEE Communications Magazine*, vol. 55, no. 6, pp. 110–117, 2017.
- [34] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, “The effects of loss and latency on user performance in unreal tournament 2003®,” *Proceedings of the ACM SIGCOMM Workshop on Network and System Support for Games, NetGames'04*, no. May 2002, pp. 144–151, 2004.

- [35] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling Large-scale, High-speed, Peer-to-peer Games," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 389–400, aug 2008. [Online]. Available: <http://doi.acm.org/10.1145/1402946.1403002>
- [36] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A Distributed Architecture for Online Multiplayer Games," in *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, p. 12. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1267680.1267692>
- [37] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," *Computer Communication Review*, vol. 34, no. 4, pp. 353–366, 2004.
- [38] S. Biswas, J. Bicket, E. Wong, R. Musaloiu-E, A. Bhartia, D. Aguayo, and C. Meraki, "Large-scale measurements of wireless network behavior," *SIGCOMM 2015 - Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 153–165, 2015.
- [39] E. Buyukkaya, M. Abdallah, and G. Simon, "A survey of peer-to-peer overlay approaches for networked virtual environments," *Peer-to-Peer Networking and Applications*, vol. 8, no. 2, pp. 276–300, 2013.
- [40] E. Carlini, A. Lulli, and L. Ricci, "Model driven generation of mobility traces for distributed virtual environments with TRACE," *Concurrency Computation*, vol. 30, no. 20, pp. 1–15, 2018.
- [41] V. Cerf and J. Postel, "Specification of Internetwork Transmission Control Program RFC 675," *INWG Protocol Note*, vol. 72, no. January, 1978. [Online]. Available: <https://tools.ietf.org/html/rfc675>http://www.cs.utexas.edu/~chris/DIGITAL/_ARCHIVE/TCPIP/IEN21.pdf
- [42] M.-C. Chan, S.-Y. Hu, and J.-R. Jiang, "An efficient and secure event signature (EASES) protocol for peer-to-peer massively multiplayer online games," *Computer Networks*, vol. 52, no. 9, pp. 1838–1845, 2008. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1389128608000960>

- [43] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X. Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005.
- [44] J. F. Chen, W. C. Lin, T. H. Chen, and S.-y. Hu, "A forwarding model for voronoi-based overlay network," *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, vol. 2, pp. 1–7, 2007.
- [45] K. T. Chen, C. Y. Huang, P. Huang, and C. L. Lei, "An empirical evaluation of TCP performance in online games," *International Conference on Advances in Computer Entertainment Technology 2006*, 2006.
- [46] M. Chen, W. Saad, and C. Yin, "Virtual reality over wireless networks: Quality-of-service model and learning-based resource management," *IEEE Transactions on Communications*, vol. 66, no. 11, pp. 5621–5635, 2018.
- [47] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," *2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016*, pp. 1–7, 2016.
- [48] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, 2002.
- [49] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols David," *SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 106–114, 1988.
- [50] J. Cloud, D. Leith, and M. Medard, "Network Coded TCP (CTCP) Performance over Satellite Networks," *Arxiv preprint*, p. 4, 2013. [Online]. Available: <http://arxiv.org/abs/1310.6635>
- [51] C. Diot and L. Gautier, "A Distributed Architecture for Multi-player Interactive Applications on the Internet," *IEEE Network*, no. August, 1999.
- [52] K. Edeline, M. Kühlewind, B. Trammell, E. Aben, and B. Donnet, "Using UDP for Internet Transport Evolution," *CoRR*, vol. abs/1612.0, 2016. [Online]. Available: <http://arxiv.org/abs/1612.07816>

- [53] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward Low-Latency and Ultra-Reliable Virtual Reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018.
- [54] H. A. Engelbrecht and J. S. Gilmore, "Pithos: Distributed storage for massive multi-user virtual environments," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 13, no. 3, 2017.
- [55] J. Fabroa, "Smilegate Entertainment announces details for the CFS 2018 Grand Finals | Business Wire," 2018. [Online]. Available: <https://www.businesswire.com/news/home/20180907005238/en/Smilegate-Entertainment-announces-details-CFS-2018-Grand>
- [56] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Y. Zomaya, "NC-CELL: Network coding-based content distribution in cellular networks for cloud applications," *2014 IEEE Global Communications Conference, GLOBECOM 2014*, pp. 1205–1210, 2015.
- [57] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 5, pp. 8–23, 1994.
- [58] C. Fogg, "What is MPEG?" 1996. [Online]. Available: https://web.archive.org/web/20090220062554/http://bmrc.berkeley.edu/research/mpeg/faq/mpeg2-v38/faq_v38.html#tag40
- [59] C. Fragouli and E. Soljanin, "Network Coding Fundamentals," *Foundations and Trends® in Networking*, vol. 2, no. 1, pp. 1–133, 2006.
- [60] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP performance," *IEEE Transactions on Mobile Computing*, vol. 4, no. 2, pp. 209–221, 2005.
- [61] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.
- [62] S. Gheorghiu, L. Lima, A. L. Toledo, J. Barros, and M. Médard, "On the performance of network coding in multi-resolution wireless video streaming," *2010 IEEE International Symposium on Network Coding, NetCod 2010*, 2010.

- [63] J. S. Gilmore, "A state management and persistency architecture for peer to peer massively multi user virtual environments," Ph.D. dissertation, 2013. [Online]. Available: <https://scholar.sun.ac.za/handle/10019.1/80268>
- [64] C. Griwodz and P. Halvorsen, "The fun of using TCP for an MMORPG," *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, 2006.
- [65] F. G. Hamza-Lup, J. P. Rolland, and C. E. Hughes, "A Distributed Augmented Reality System for Medical Training and Simulation," *CoRR*, vol. abs/1811.1, pp. 1–18, nov 2018. [Online]. Available: <http://arxiv.org/abs/1811.12815>
- [66] J. He, J. Yang, C. An, H. Liu, and X. Li, "Analysis on MAC layer retransmission scheme in wireless networks," *MobiWac 2008 - Proceedings of the 6th ACM International Symposium on Mobility Management and Wireless Access*, pp. 161–165, 2008.
- [67] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.
- [68] S.-y. Hu, "Spatial Publish Subscribe," in *IEEE Virtual Reality (IEEE VR)*, 2009, p. 6. [Online]. Available: <http://pap.vs.uni-due.de/MMVE09/papers/p8.pdf>
- [69] S.-y. Hu, J. F. Chen, and T. H. Chen, "VON: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, jul 2006.
- [70] S.-y. Hu and K. T. Chen, "VSO: Self-organizing spatial publish subscribe," *Proceedings - 2011 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2011*, pp. 21–30, 2011.
- [71] S.-y. Hu, C. Wu, E. Buyukkaya, C. H. Chien, T. H. Lin, M. Abdallah, J. R. Jiang, and K. T. Chen, "A spatial publish subscribe overlay for massively multiuser virtual environments," *ICEIE 2010 - 2010 International Conference on Electronics and Information Engineering, Proceedings*, vol. 2, no. Iceie, pp. V2–314–V2–318, 2010.
- [72] International Electrotechnical Commission, "IEC 60063:2015 Preferred number series for resistors and capacitors." [Online]. Available: <https://webstore.iec.ch/publication/22011>

- [73] Jana Iyengar (Fastly) and Martin Thomson (Mozilla), “QUIC: A UDP-Based Multiplexed and Secure Transport draft-ietf-quic-transport-17,” 2018. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-quic-transport-24>
- [74] D. R. Jefferson, “Virtual Time,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985.
- [75] J. R. Jiang, Y. L. Huang, and S.-y. Hu, “Scalable AOI-cast for peer-to-peer networked virtual environments,” *Journal of Internet Technology*, vol. 10, no. 2, pp. 119–125, 2009.
- [76] E. E. Johnson, M. Balakrishnan, and Z. Tang, “Impact of turnaround time on wireless MAC protocols,” *Proceedings - IEEE Military Communications Conference MILCOM*, vol. 1, no. C, pp. 375–381, 2003.
- [77] S. Katti, M. Muriel, and J. Crowcroft, “XORs in the Air: Practical Wireless Network Coding,” no. September 2006, pp. 2–5.
- [78] Y. Kawahara, T. Aoyama, and H. Morikawa, “A Peer-to-Peer Message Exchange Scheme for Large-Scale Networked Virtual Environments,” *Telecommunication Systems*, vol. 25, pp. 353–370, 2002.
- [79] S. A. Khan, M. Moosa, F. Naeem, M. H. Alizai, and J. M. Kim, “Protocols and Mechanisms to Recover Failed Packets in Wireless Networks: History and Evolution,” *IEEE Access*, vol. 4, no. July, pp. 4207–4224, 2016.
- [80] S. Kulkarni, “Badumna Network Suite: A decentralized network engine for Massively Multiplayer Online applications,” in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, 2009, pp. 178–183.
- [81] J. Le, J. C. Lui, and D. M. Chiu, “DCAR: Distributed coding-aware routing in wireless networks,” *IEEE Transactions on Mobile Computing*, vol. 9, no. 4, pp. 596–608, 2010.
- [82] J. Lee, H. Lee, S. Ihm, T. Gim, and J. Song, “Apolo: Ad-hoc peer-to-peer overlay network for massively multi-player online games,” Tech. Rep., 2005. [Online]. Available: <https://www.cs.kaist.ac.kr/upload{ }files/report/1227157612.pdf>
- [83] H. Liang, R. Nilaksha, W. Tsang, and O. Mehul, “Avatar mobility in user-created networked virtual worlds : measurements , analysis , and implications,” pp. 163–190, 2009.

- [84] F. Liu, Z. Chen, and B. Xia, "Data Dissemination With Network Coding in Two-way Vehicle-to-vehicle Networks," vol. 65, no. 4, pp. 2445–2456, 2016.
- [85] H. Liu and Y. Lo, "DaCAP - A Distributed Anti-Cheating Peer to Peer Architecture for Massive Multiplayer On-line Role Playing Game," in *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, may 2008, pp. 584–589.
- [86] M. G. Luis, "TCPDUMP/LIBPCAP public repository," *Online document*, 2009. [Online]. Available: <https://www.tcpdump.org/>
- [87] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents RFC 7414." [Online]. Available: <https://tools.ietf.org/html/rfc7414{#}section-4.4>
- [88] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: A Network Software Architecture for Large Scale Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 3, no. 4, pp. 265–287, 1994. [Online]. Available: <https://doi.org/10.1162/pres.1994.3.4.265>
- [89] D. J. C. MacKay, "Fountain codes," *IEEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.
- [90] M. Meehan, M. C. Whitton, and F. P. Brooks, "Effect of Latency on Presence in Stressful Virtual Environments," *IEEE Virtual Reality*, vol. 2003, 2003.
- [91] F. P. Miller, A. F. Vandome, and J. McBrewster, *Hamming Code: Parity Bit, Two-Out-Of-Five Code, Hamming(7,4), Reed-Muller Code, Reed-Solomon Error Correction, Turbo Code, Low-Density Parity- Check Code, Telecommunication, Linear Code*. Alpha Press, 2009.
- [92] P. Morillo, W. Moncho, J. M. Orduña, and J. Duato, "Providing Full Awareness to Distributed Virtual Environments Based on Peer-to-peer Architectures," in *Proceedings of the 24th International Conference on Advances in Computer Graphics*, ser. CGI'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 336–347. [Online]. Available: <http://dx.doi.org/10.1007/11784203{ }29>

- [93] J. Müller and H. Sergei Gorlatc, "Rokkatan: Scaling an RTS Game Design to the Massively Multiplayer Realm," *ACM Computers in Entertainment (CIE)*, vol. 4, no. 3, pp. 1–14, 2006. [Online]. Available: <http://dx.doi.org/10.1145/1146816.1146833>
- [94] K. Nguyen, T. Nguyen, and S. C. Cheung, "Peer-to-peer streaming with hierarchical network coding," *Proceedings of the 2007 IEEE International Conference on Multimedia and Expo, ICME 2007*, pp. 396–399, 2007.
- [95] C. Partridge, J. Hughes, and J. Stone, "Performance of checksums and CRCs over real data," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM 1995*, pp. 68–76, 1995.
- [96] M. V. Pedersen, J. Heide, and F. H. Fitzek, "Kodo: An open and research oriented network coding library," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6827 LNCS, 2011, pp. 145–152.
- [97] A. Petlund, "Improving latency for interactive, thin-stream applications over reliable transport," Ph.D. dissertation, 2009.
- [98] A. Petlund, K. Evensen, C. Griwodz, and P. Halvorsen, "TCP enhancements for interactive thin-stream applications," *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 127–128, 2008.
- [99] N. Pino, "Valve Index review | TechRadar." [Online]. Available: <https://www.techradar.com/reviews/valve-index>
- [100] A. Ploss, S. Wichmann, F. Glinka, and S. Gorlatch, "From a single-to multi-server online game: A quake 3 case study using RTF," *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, ACE 2008*, no. January 2008, pp. 83–90, 2008.
- [101] J. Postel, "RFC 768 - User Datagram Protocol," pp. 1–3, 1980. [Online]. Available: <https://tools.ietf.org/html/rfc768>
- [102] M. C. Potter, B. Wyble, C. E. Hagmann, and E. S. McCourt, "Detecting meaning in RSVP at 13 ms per picture," *Attention, Perception, & Psychophysics*, vol. 76, no. 2, pp. 270–279, feb 2014. [Online]. Available: <https://doi.org/10.3758/s13414-013-0605-z>

- [103] R. Prior and A. Rodrigues, "Systematic network coding for packet loss concealment in broadcast distribution," *International Conference on Information Networking 2011, ICOIN 2011*, pp. 245–250, 2011.
- [104] K. Raaen, "Response time in games : Requirements and improvements," Ph.D. dissertation, University of Oslo, 2015. [Online]. Available: <http://home.ifi.uio.no/paalh/students/KjetilRaaen-phd.pdf>
- [105] P. Rajesh Desai, P. Nikhil Desai, K. Deepak Ajmera, and K. Mehta, "A Review Paper on Oculus Rift-A Virtual Reality Headset," *International Journal of Engineering Trends and Technology*, vol. 13, no. 4, pp. 175–179, 2014.
- [106] S. Sarkar, "Blizzard reaches 100M lifetime World of Warcraft accounts," 2014. [Online]. Available: <https://www.polygon.com/2014/1/28/5354856/world-of-warcraft-100m-accounts-lifetime>
- [107] G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis, "Requirements of Peer-to-Peer-based Massively Multi-player Online Gaming," *Seventh IEEE International Symposium on Cluster Computing and the Grid CCGrid 07*, pp. 773–782, 2007.
- [108] F. Schmidt, M. H. Alizai, I. Aktas, and K. Wehrle, "Reflector: Heuristic header error recovery for error-tolerant transmissions," *Proceedings of the 7th Conference on Emerging Networking EXperiments and Technologies, CoNEXT'11*, 2011.
- [109] H. Seferoglu, A. Markopoulou, and K. K. Ramakrishnan, "I 2 NC: Intra-and inter-session network coding for unicast flows in wireless networks," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 1035–1043.
- [110] R. K. Sheshadri and D. Koutsonikolas, "On packet loss rates in modern 802.11 networks," *Proceedings - IEEE INFOCOM*, 2017.
- [111] A. Shokrollahi, "An Introduction to Low-Density Parity-Check Codes," in *Theoretical Aspects of Computer Science*. Springer, Berlin, Heidelberg, 2002, ch. 1, pp. 175–197.
- [112] L. Simic, J. Riihijarvi, and P. Mahonen, "Measurement study of IEEE 802.11ac Wi-Fi performance in high density indoor deployments: Are wider channels always better?" *18th IEEE*

International Symposium on A World of Wireless, Mobile and Multimedia Networks, WoWMoM 2017 - Conference, 2017.

- [113] SUTHERLAND IE, "Head-Mounted Three Dimensional Display," vol. 33, no. pt 1, pp. 757–764, 1968.
- [114] D. Szabó, A. Csoma, P. Megyesi, A. Gulyás, and F. H. P. Fitzek, "Network coding as a service," *Infocommunications Journal*, vol. 7, no. 4, pp. 2–11, 2015. [Online]. Available: <https://arxiv.org/pdf/1601.03201.pdf>
- [115] D. Szabo, A. Gulyas, F. H. P. Fitzek, and D. E. Lucani, "Towards the Tactile Internet: Decreasing Communication Latency with Network Coding and Software Defined Networking," *European Wireless 2015; 21th European Wireless Conference; Proceedings of*, pp. 1–6, 2015.
- [116] H. Taylor, "Dungeon Fighter Online gross lifetime revenue exceeds \$10bn | GamesIndustry.biz," 2018. [Online]. Available: <https://www.gamesindustry.biz/articles/2018-06-21-dungeon-and-fighter-gross-lifetime-revenue-exceeds-usd10bn>
- [117] P. Vingelmann, M. V. Pedersen, F. H. P. Fitzek, and J. Heide, "Multimedia distribution using network coding on the iphone platform," in *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing - MCMC '10*, 2010, p. 3. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1877953.1877957>
- [118] C. Westphal, "Challenges In Networking To Support Augmented Reality And Virtual Reality," *Icnc 2017*, 2017. [Online]. Available: <https://users.soe.ucsc.edu/~cedric/papers/westphal2017challenges.pdf>
- [119] A. D. Wyner and S. Shamai, "Introduction to "Communication in the presence of noise" by C. E. Shannon," *Proceedings of the IEEE*, vol. 86, no. 2, pp. 442–446, 1998.
- [120] L. F. Xie, P. H. J. Chong, and Y. L. Guan, "Performance Analysis of Network Coding With Virtual Overhearing in Wireless Networks," vol. 64, no. 5, pp. 2051–2061, 2015.
- [121] A. Yahyavi and B. Kemme, "Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey," *ACM Computing Surveys*, vol. 46, no. 1, pp. 434–442, 2013.
- [122] T. Yamada, "Block Codes," in *Essentials of Error-Control Coding Techniques*, 1990, pp. 39–59.

- [123] A. Zeb, M. Asim, W. Ali, and N. Mufti, "Performance analysis of WLAN in the presence of co-frequency microwaves," in *International Conference on Communication Technologies, ComTech 2017*, 2017, pp. 7–11.
- [124] B. Zhang, Z. Liu, S. H. Chan, and G. Cheung, "Collaborative Wireless Freeview Video Streaming with Network Coding," *IEEE Transactions on Multimedia*, vol. 18, no. 3, pp. 521–536, 2016.
- [125] M. Zhang, M. Dusi, W. John, and C. Chen, "Analysis of UDP traffic usage on internet backbone links," *Proceedings - 2009 9th Annual International Symposium on Applications and the Internet, SAINT 2009*, no. July, pp. 280–281, 2009.

Appendices

Appendix A

Test Data Acquisition

In this chapter we will briefly explain how the test data was captured. We needed to store the state of each node, latency measurements and bandwidth measurements at each timestamps. In order to achieve this, we used three log files per node, saving them to the hard drive as the test is running. We then calculated the global state consistency, latency and bandwidth after the test was completed.

A.1 Movement and node status: VASTStatLog

In order to compute the topology consistency and drift distance of each node, we needed to capture the positions and neighbour list of each node at each update step. For this we used the class `VASTStatLog`. At each VAST tick, the position of the `VASTClient` was saved in the log as well as its AOI neighbours. In order to determine the number of connected nodes, we also stored the *JOINED* state of the node as well as if it is a *Matcher*. Each tick created another step in the `VASTStatLog` record. The step is also saved with the timestamp at which is computed. Note that the ticks of the VAST executables on the node are not synchronised. However, the same wall clock is used to generate the timestamps, so that the timestamps are comparable.

In order to compute the state consistency, a bucket synchronisation method are used: the gateway node's timestamp is used as the start of the step, and all records that fall within one update interval (100 ms) of this step is considered. There is only one update per node within that window as each other node also updates at 100 ms intervals.

In order to calculate the topology consistency, the total number of known neighbours from all the clients in the `VASTStatLogs` are

Table A.1: Mean and standard deviation of 100 round-trip ping times on physical network. Results in milliseconds.

Link	Physical	
	Mean	Std dev
h1 -> h2	0.2533	0.0378
h2 -> h1	0.2608	0.0352

counted. Using the positions of each client, the number of actual neighbours are counted.

A.2 Latency: VASTLatencyStatLog

The current clock time is added to each outgoing *MOVE* message. On arriving the receiving node, the timestamp is parsed and compared to the current clock time on the receiving node. The difference is calculated and recorded. At the end of each step, the latency records of that step is stored in the VASTLatencyStatLog. Note that the number of records are variable on each node as it depends on the number of incoming messages.

In order to compute the average latency, all of the records (network wide) that fall within the timestep (as specified by the gateway's timestamp as before) are aggregated and the average is computed.

Note that due to the update focussed nature of the VAST, it is most likely that the incoming message will only be processed at the Matcher in the next update step. Instead of processing the messages continually, the receive messages in the queue are processed as a batch during each update step. The only exception to this is if the message is intended for a client or matcher on the same node; it is then processed in the same update step.

On the Mininet emulator, the clock used to provide the send time and receive time is the same clock as both hosts use the kernel clock. For the physical host setup, the Network Time Protocol [15] was used to synchronise the host clocks. The clock synchronisation is dependent on the network delay between hosts, typically NTP attempts to synchronise clocks to an accuracy within the mean latency between. The latency measurements using the `ping` utility is summarised in Table A.1.

Despite the clocks being synchronised, the steps at each node is not synchronised. Therefore the processing of message (and therefore latency) recording is not synchronised. Depending on the when updates happen on different nodes, the queue waiting time can be between 0 - 100 ms. This is a realistic representation of a distributed

system as synchronisation of update steps between nodes on the Internet would be near impossible.

To clarify clocks are synchronised in order to correctly calculate latency. Update steps are not synchronised to represent a distributed VE implementation.

A.3 NIC Send / Receive bandwidth: tshark

In order to capture the NIC send and receive bandwidth, the commandline utility `tshark` was used. `tshark` is part of the Wireshark [26] project, a set of utilities for capturing and analysing network packets.

`tshark` allows the capturing of network packets on the network interface, showing the packet type (ARP, IP, UDP or TCP), packet length in bytes, source and destination IP addresses, source and destination ports. From this information we can classify which application and destination a particular packet belongs to. Using this information, we can sum the incoming and outgoing bandwidth to the VAST application on each node.

As before, the send and receive bandwidth records are discretised according to the gateway's time steps. The total network bandwidth of all nodes are normalised by the number of active nodes in the network to give the per node NIC bandwidth.

A.4 Network Coded Bandwidth: VASTNetStatLog

In order to determine the network coded bandwidth, we recorded the number of received bytes of encoded packets in the coded pipeline. This is equivalent to measuring the sent bytes at the coding host. Similar VASTStatLog, we discretise the records into steps according to the gateway. The total network coded bandwidth is then normalised by the number of active nodes.