BURST

Bulk Transfer Optimization over High Latency Networks with UltraFast® AI/ML Technology

WHITE PAPER

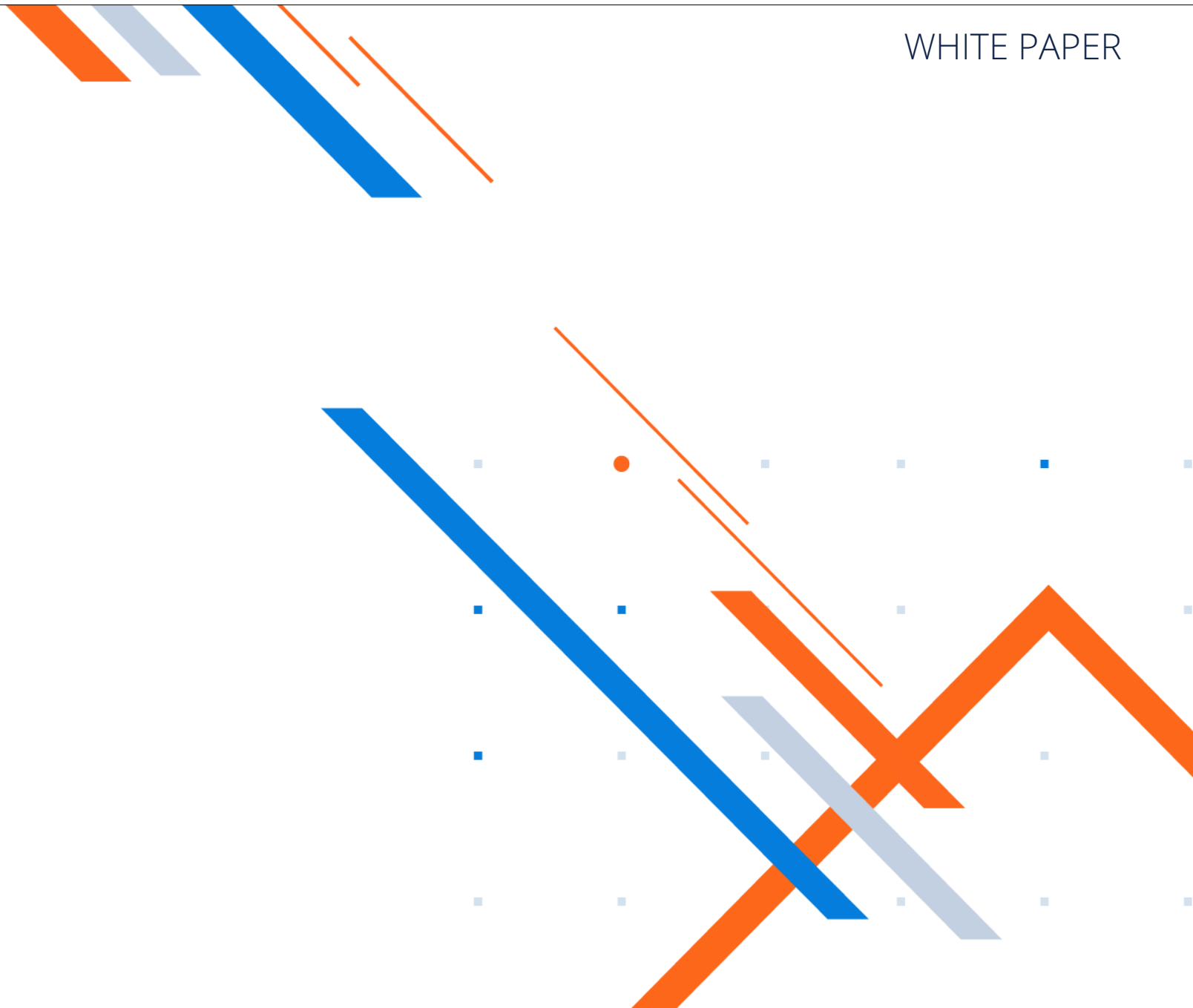


TABLE OF CONTENTS

Bulk Data Transfer Challenges 3
TCP/IP and Its Issues with Latency and Packet Loss 3
What causes Network Loss? 4
Ideal transmission 4
Lossy transmission 5
The UltraFast Solution 5
Reinforcement Learning Concepts 6
The UltraFast Gambler Agent 6
The UltraFast Reinforcement Learning Process 7
The UltraFast Architecture 8
UltraFast Speed Test Results 9
Summary 10

Bulk Transfer Optimization over High Latency Networks with UltraFast® AI/ML Technology

Bulk Data Transfer Challenges

Enterprises and agencies generate large amounts of data, often in the terabytes to many hundreds of terabytes or more, in various edge locations, remote offices, factories, datacenters and clouds. This data often needs to be transferred from where it is generated or stored to other locations, often for analytics, disaster recovery and backup, and increasingly, to train and run machine learning applications. The weak link between these diverse locations is often the network. Networks suffer from various challenges, including oversubscription/overload from too much traffic at times (e.g., during backup jobs or other high demand periods). Certain types of networks like satellite and cellular involve high latency due to the distances and technologies involved to access remote sites.

These network limitations, coupled with high demand for large or bulk data transfers, result in many circumstances where the remote data simply cannot be transferred over these network conditions. Instead, expensive and painful workarounds, such as personnel having to travel to each remote location, manually copy data onto USB drives, then manually transport or ship these data devices from one location to another.

Customers need to automate bulk data transfers and make use of whatever network connections are available at each location, despite the network challenges around latency, packet loss and/or intermittent connectivity.

The most common challenges stem from latency and packet loss, which is where we will focus in this paper.

TCP/IP and Its Issues with Latency and Packet Loss

For good reason, TCP is the standard protocol used today to ensure reliable duplex connection over the Internet. TCP is known for its reliability over low latency, low loss networks. It transmits in a consistent, predictable manner, perfect for planning typical IT implementations, either in-house or over relatively short distances.

However, TCP does not perform well when the network is congested or when the points of contact are at a distance resulting in greater than 150/200 milliseconds of latency. The reason for these drawbacks is the inherent way in which the TCP transmitter manages the transmission. For each packet loss from the source to the destination, the transmitter automatically decreases the transmission window to a lower bandwidth option to ensure successful packet delivery.

Most accelerators, either in software or hardware, simulate a transmission window that is not related to the perceived loss but to some estimate of the actual loss within the network. This acceleration method allows the sender to send without being blocked by a random loss; however, this means that the receiver must store the received packets in memory until the loss is recovered to maintain the order of the transmission for the receiver.

The loss of a given network without the current transmission will be referred to in the current document as “loss zero.” This “loss zero” estimate will be vital to the design of our UltraFast accelerator.

What causes Network Loss?

There are numerous potential reasons for the loss of a packet in a network. However, the most common reason for network loss, apart from hardware problems, is the congestion avoidance method implemented in normal Internet routers. This congestion avoidance method is called RED (Random Early Detection). There are variants to how RED handles congestion avoidance, but usually, they fall into the basic category of randomly dropping a packet when the router/switch senses that its queues are filling up. The receiver will then notice the packet loss, and the sender (most often using TCP, of course) will lower its window to accommodate for the perceived congestion.

This method is highly effective but overly aggressive. It penalizes the transmission too much, as the same result would be realized if the sender noticed the loss and retransmitted the lost packet. Instead, the sender has to deal with the altered transmission window for the remainder of the transmission.

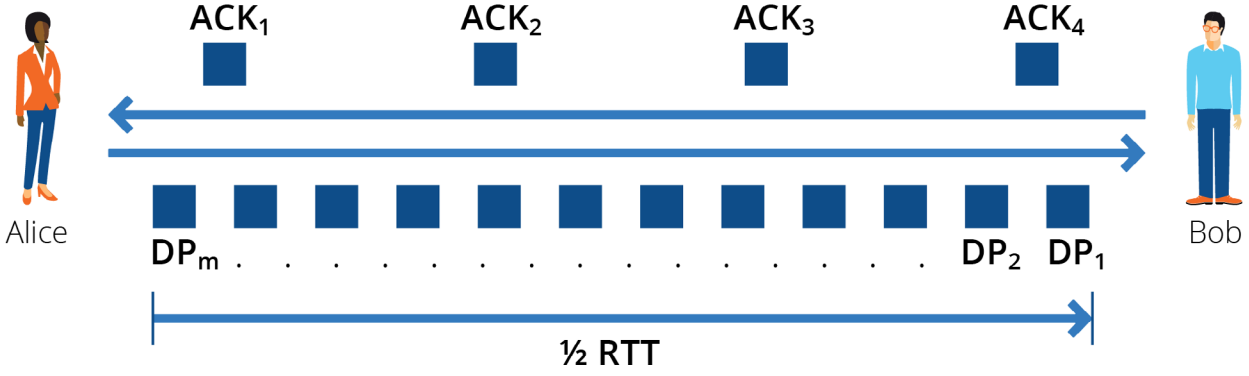
In any case, to exploit as much of the available bandwidth as possible, a typical accelerator software component must try to probe the network. When probing the network, it must accurately estimate when the network is at the edge of congestion and avoid surpassing it, or the losses will lower the transmission rate exponentially.

Ideal transmission

In an ideal transmission between A and B, the transmitter will synchronize its transmission rate based on an objective measure of the physical network model, as depicted in *Pic. 1*.

As you can see, Alice has synchronized her transmission perfectly with the network. She has "filled the wire" with packets so well that Bob will immediately receive a packet as soon as a packet is placed into the wire. Bob does not experience any packet loss.

He sends periodic ACK packets back to Alice, which signals to Alice that she can continue to send packets at the same rate.



(Pic. 1)

Lossy transmission

Lossy transmission rate is depicted in *Pic. 2*.

Between our imaginary Alice and Bob, there is often a router. Usually, there are many more routers than just one, and each with its own capacity. Across the unconditioned Internet networks, routes are unpredictable and subject to constant change, traffic spikes from competing network users, and router overruns.

These routers might connect two networks with different bandwidths and delays. For example, from Alice to the router, a high-speed network operates, while from the router to Bob, we find a slower one.

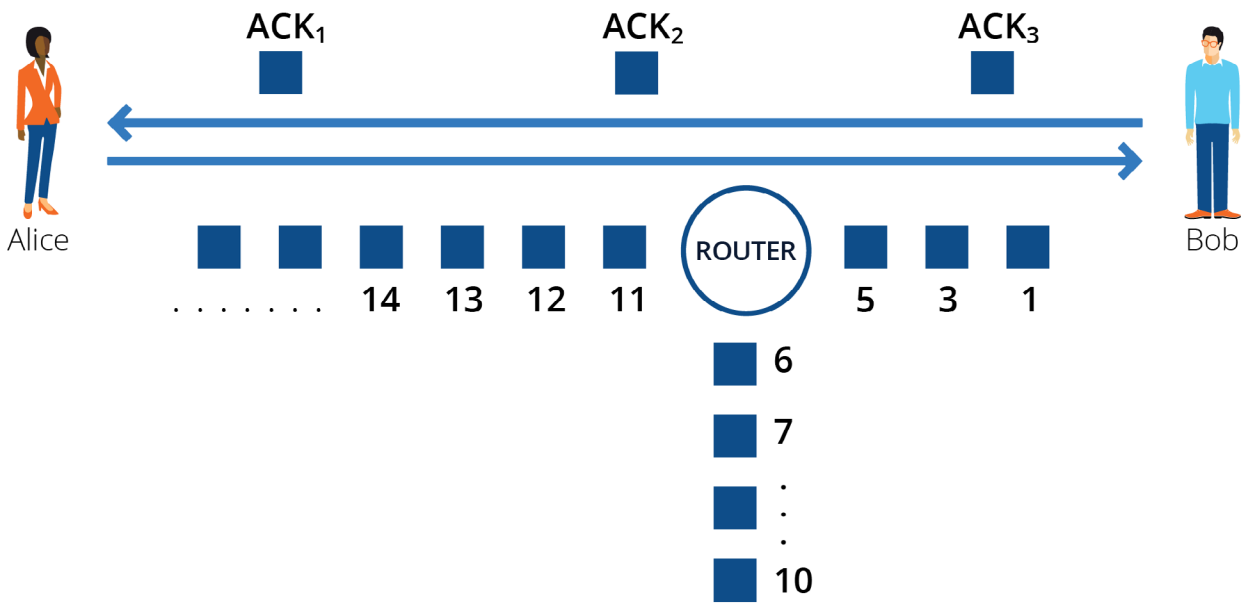
When this occurs, the router on the lower-speed network will be forced to accumulate packets until its queues reach a certain threshold. At this point, it will start to drop packets because it needs to maintain a certain amount of free space in its queue to avoid a complete network stall. As we explained earlier, loss of packets results in throttling for TCP/IP transmissions. In essence, the slower router determines the pace of transmission, often undercutting the true bandwidth available significantly.

The UltraFast Solution

UltraFast takes a vastly different approach to address high-latency, lossy networks than TCP:

1. It calculates a "loss zero" value by observing reference packets, "probe" packets that are sent across the network on fixed intervals to measure network stability, latency, and losses
2. It uses a type of machine learning known as "reinforcement learning" to learn the actual network characteristics and dynamically adapt to changing network conditions, including round trip time (RTT), capacity, and data transmission successes vs. losses.

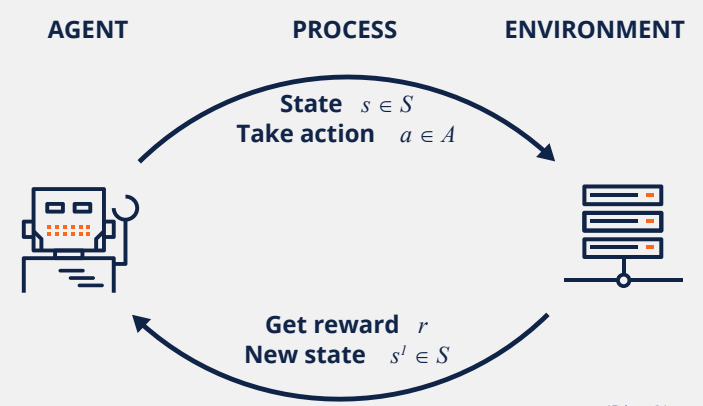
Let's first understand the basics of reinforcement learning before diving deeper into UltraFast's design.



(Pic. 2)

Reinforcement Learning Concepts

In Reinforcement Learning, an Agent interacts with the Environment (in our case, a complex network), running experiments known as Actions using a given State, as shown in *Pic. 3*. Each Action interacts with the environment, and the results of those interactions are recorded. Those results are then evaluated based upon a Policy to determine whether to Reward or Penalize each experiment. The Agent then determines the best new State and repeats the process with a new set of experiments.



(Pic. 3)

As shown above, "The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many states (s ∈ S) of the environment and choose to take one of many actions (a ∈ A) to switch from one state to another. Which state the agent will arrive in is decided by transition probabilities between states. Once an action is taken, the environment delivers a reward (r ∈ R) as feedback."[1]

The learning Agent continuously runs experiments, collects results, and over time converges upon the optimal or desired State. In effect, the Agent runs experiments, observes results, adjusts the next experiment based upon its learnings, and gets better with each set of experiments until the desired results are achieved, dynamically converging upon the ideal outcome it seeks.

Unlike supervised learning that uses neural networks, there is no training data set needed or used to train a model. Some reinforcement learning algorithms are actually model-free; I.e., there is no prior knowledge of the environment available ahead of time; instead, the model is created as state information based upon observations made about each experimental action taken over time.

[1] Weng, L. A (Long) Peek into Reinforcement Learning. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

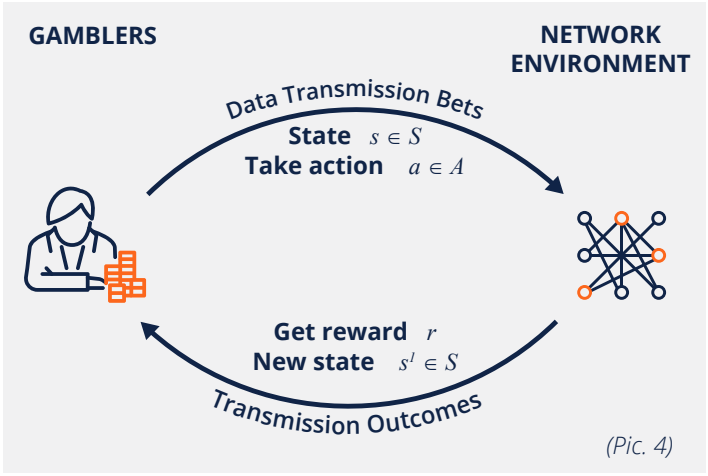
The UltraFast Gambler Agent

UltraFast uses a machine learning process that uses a set of "gamblers," or data transmission experiments, that place different "bets" on what the ideal transmission rate will be. There is no model of the network available ahead of the agent running its own experiments to learn about the network.

The main goals are to:

1. Maximize network throughput by sending as much data as possible
2. Avoid creating packet loss due to sending data too quickly
3. Detect when external factors, such as other network participants, changing IP routes, and other dynamic conditions are causing congestion or interfering with packet throughput and use this information to place improved bets.

Pic. 4 depicts the UltraFast reinforcement learning process.



(Pic. 4)

The Agent creates a set of "Gambler" processes, each running in an independent thread. Gamblers are given a "Data Transmission Bet" to place, said bet being its data transmission "rate"; i.e., a bet is the time delay between sending each packet. The data is sent to a

connection at the distant end of the network, and several types of responses may occur:

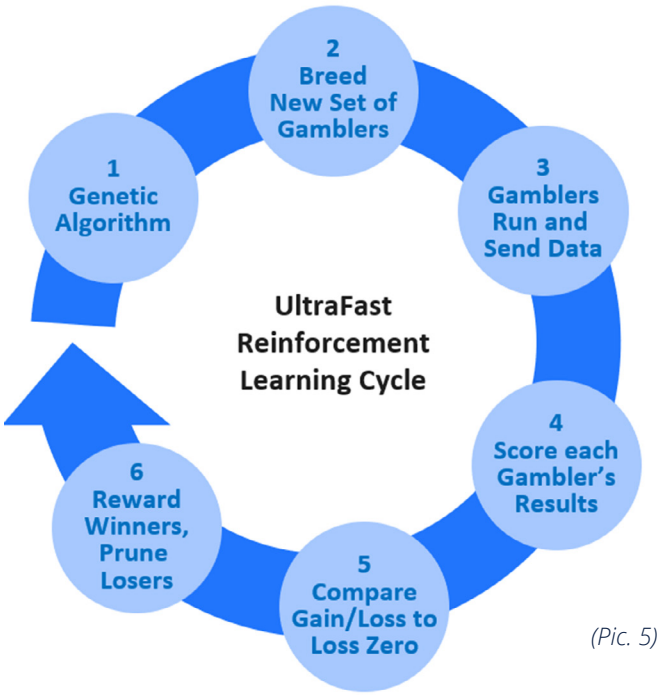
1. an ACK indicating good data receipt,
2. a NAK indicating bad data receipt, or
3. no response at all, indicating a lost packet (timeout).

Each Gambler process sends several data packets and records the overall success rate, i.e., how many packets were sent, how many succeeded, and how many failed. Upon completion of the Gamblers' processing, each Gambler is assigned an overall score. The more acknowledged and successful data packets sent, the higher the score. The more NAKs or timeouts (packet losses) present, the lower the score.

As we will soon see in more detail, the Agent then uses these Gambler scores to reward successful gamblers, which are allowed to "breed" and multiply during the next generation or experiment cycle. Less successful or failed Gamblers are pruned and eliminated. This process is similar to natural selection, where the strong and successful survive, and the weak and unsuccessful do not propagate.

The UltraFast Reinforcement Learning Process

Pic. 5 depicts the UltraFast learning cycle and each step in the process.



(Pic. 5)

The UltraFast learning loop runs repeatedly, processing these steps:

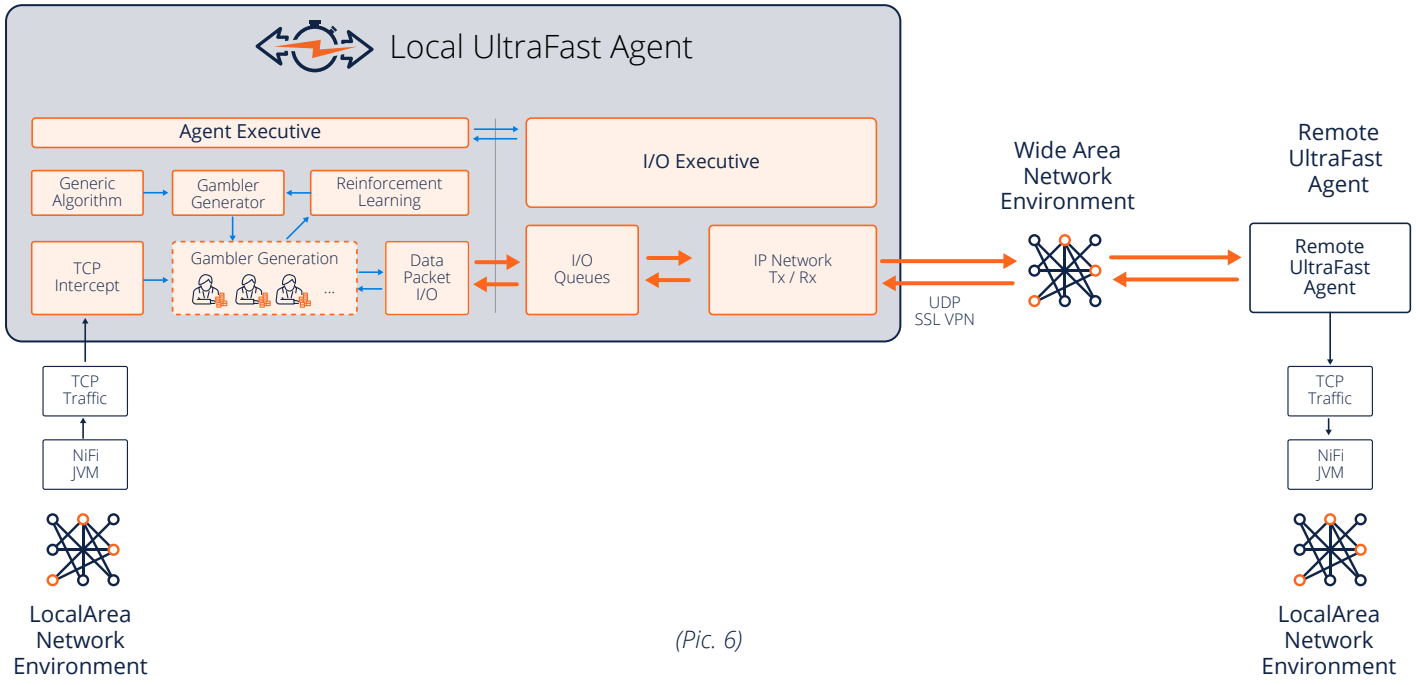
1. A Monte Carlo derived genetic algorithm generates random strategies for the initial set of gamblers, then subsequently breeds new gamblers based upon last cycle's winners' results.
2. A new generation of dozens of gamblers is created at the start of each cycle, each with its own rate of sending data packets.
3. Gamblers send their data packets, measuring ACKs, NAKs, and lost packets.
4. Each gambler's win/loss rate is scored – more packets sent equals a higher score, lost packets or data transmission errors (NAKs) penalize the score.
5. Each gambler's loss-rate is compared with the current loss-zero (separately established with regularly timed packets).
6. Winning gamblers showing the best results are rewarded by being bred, resulting in similar successful gamblers for the next cycle. The agent prunes the losers and feeds the learned results forward into the genetic algorithm. In addition to the 'successful' newly created gamblers, new random variants are added to further explore the newly defined boundaries, enabling the system to adapt to changing network conditions.

The above 6-step process runs continually, optimizing data throughput while minimizing packet loss and congestion, and adapting to the constantly changing and evolving complex network environment.

Reinforcement learning enables UltraFast to adapt to each unique network topology and navigate its evolving traffic and routing conditions.

The UltraFast Architecture

Pic. 6 architecture diagram shows how UltraFast is organized and operates at a high level.



(Pic. 6)

The Local UltraFast Agent is comprised of two main execution units:

1. The Agent Executive, which is responsible for the data transmission strategy using reinforcement learning and a loss-zero reference
2. The I/O Executive, which handles network data buffering, sequencing, and input/output via the operating system's network stack.

Data flows into the UltraFast Agent as intercepted TCP traffic, often from Apache NiFi processes running on a Java Virtual Machine (JVM), in the case of the Fuusion product. This traffic is typically an SSL-encrypted data stream. The intercepted data is queued temporarily (not shown), then fed through each gambler process, which queues data packets to the I/O Executive. The I/O executive sends UDP over IP packets, which can optionally be further encapsulated by a VPN tunnel, such as an SSL/VPN (which adds some overhead but additional security and a wrapper for UDP).

The aforementioned reinforcement learning cycle ensues and transmits the data and converges upon the ideal data transmission rate(s), as described above.

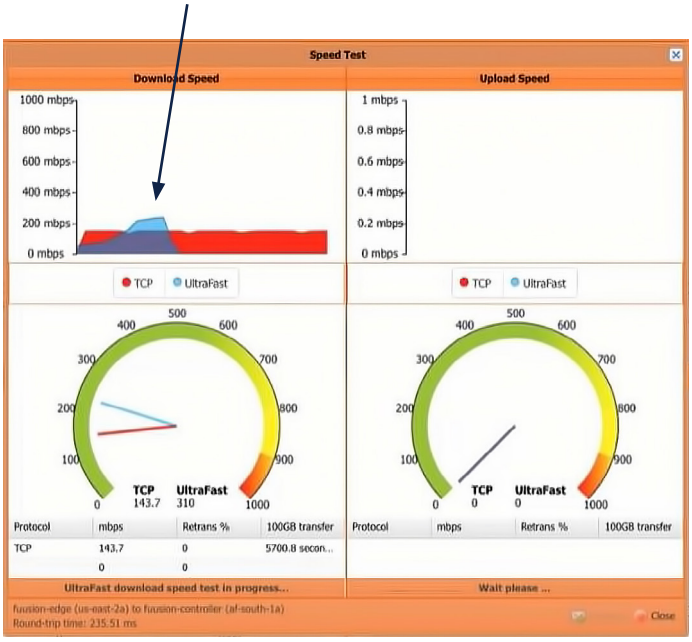
The Remote UltraFast Agent receives the data packets, sends ACK packets in response as good packets are received (or NAKs for bad or failed packets). It also handles reflecting the "probe" packets (not shown) back to the sender. The agent also ensures the data packets are ordered in the correct, original sequence. Finally, the reassembled data stream exits UltraFast as TCP traffic arriving on the remote LAN, where it can be consumed by a remote process port (NiFi) or other network destination(s), as appropriate.

UltraFast Speed Test Results

The following screenshot shows the results from running the UltraFast Speed Test feature. Similar to other network download speed tests used to examine Internet download/upload speed, this test runs an "iperf" data stream through both TCP/IP and UltraFast to compare the difference in results from each.

In *Pic. 7*, we see a download test run with TCP/IP shown in red. The average TCP/IP throughput was 143.7 Mbps at 235 milliseconds of network round trip time. This particular test run was between AWS Ohio region and AWS Capetown, South Africa. Note there was no packet loss recorded; otherwise, TCP/IP's throughput would have been much worse.

UltraFast Learning Ramp



(Pic. 7)

The light blue chart shows UltraFast, which is starting its reinforcement learning process, having been through several generations of gambler learning cycles at this point. It is learning the network characteristics and ramping up to optimize for this high-latency network.

Pic. 8 shows the outcome of the completed speed test. We can clearly see a tipping point is reached, whereby UltraFast learns the optimal data transmission rate and can make full use of the available 1 Gbps network, despite the latencies involved. Towards the end of the test, a network impact occurred, causing UltraFast to readjust its policy and adapt to the new conditions.

(Pic. 8)

Interestingly, in the subsequent Upload speed test, UltraFast has already learned how to optimize for this network. The upload speed is optimal from the start, averaging about 822 Mbps vs. TCP's 144 Mbps, or 5.7 times (5700%) more throughput. Keep in mind this test is over AWS peering network across regions, which is typically a very stable, clean-conditioned network, unlike many WAN's and typical satellite networks, cellular networks, and other edge and remote location networks.

Summary

Remote locations utilize networks often challenged by latency and packet loss. The need to transfer large amounts of data to and from remote and edge locations is increasingly challenging with the growth of edge computing and IoT. The network becomes the weak link between the edge and remote locations and centralized data centers, including public and private clouds.

We have seen how UltraFast's unique approach to network optimization uses a patented combination of regularly-spaced probe packets to determine loss-zero of the network, along with reinforcement learning to adapt to each network's unique and changing network environment conditions.

UltraFast overcomes the limitations imposed by TCP/IP for bulk data transmission over high-latency and lossy networks. We have seen typical network throughput performance increases ranging from 5 times faster on up to 100 times faster in networks with both high-latency and significant packet loss due to congestion. These conditions are typical of WANs, satellite links, cellular and packet radio networks, and other networks connecting with remote locations and edge devices (e.g., DSL).

For more details about UltraFast and its patented technology, please refer to Buurst patents:

Original Patents: 9,967,198

Continuation Patent No. 10,057,176

UltraFast is a feature of the Fuusion product. For more information on how to leverage UltraFast technology to address Edge-to-Cloud, remote data transfer, and migration use cases, please refer to the Buurst Fuusion website.