
CHAPTER 3

Prior Work

The Mirage model evolved from concepts from a variety of disciplines, and is also a paradigm for latency in communication. As such, the relevant prior work spans a breadth of disciplines in traditional protocol research and communication models, as well as in distributed systems, cybernetics, and physics. This is a summary of that prior work and a discussion of Mirage in comparison.

This chapter discusses prior work in general protocols and communication, rather than as it relates to specific applications of the model. Discussion of prior work relevant to these particular applications is presented in the chapter where the Mirage model is applied to specific protocols (the Network Time Protocol, Chapter 4) or systems (processor-memory interaction in the μ -Net evaluation, Chapter 6). Elaborations of particular similarities are presented in appendices, e.g., relevance to Shannon's model of communication (Appendix A), equivalences in timed Petri Nets (Appendix F), and original conceptions from analogies in physics (Appendix B).

Mirage is also related to some recent high speed optimizations of protocol implementations. Most of these optimizations are designed to reduce computational requirements of high speed protocols, rather than to compensate for increased effects of bit latency, as Mirage is intended. In hindsight, these optimizations can be viewed as

versions of some aspects of communicability in Mirage, although none has been derived or presented as such.

Mirage is a system comprised of sender modeling of a receiver, sender anticipation, entropic stability, and guarded messages. All of these components are intended to describe latency compensation if the information in transit exceeds the determinism of the system. Whereas some of these components appear in prior research, their combination in Mirage provides a novel system for understanding the effects of latency in communication.

Mirage originally evolved from analogs to principles in physics, most notably those of imprecision of state, quantum interaction by the exchange of field particles, and Feynman path integrals. These analogies are addressed further in Appendix B, so as not to confuse the conceptual basis for Mirage with its development and current design as a model.

3.1. Prior models of communication

There are a few canonical finite state machine (FSM) models used in protocol analysis; these include Shannon's model of communication, communicating FSMs, Petri Nets (and their variants), and language-based descriptions such as LOTOS and Estelle [Si91], [Ve86], [Sc82a]. Mirage differs from these models primarily in its use of partitions for communicability, although individual models individually exhibit similarities to components of Mirage.

3.1.1. The models

The Mirage model is most similar to the anticipatory feedback mechanisms of a compensator in the presence of effector delays, from cybernetics theory [Wi48]. This work describes the notion of sender (compensator) modeling of receiver (effector) state to maintain stability. This modeling is the converse of that of Shannon's model of communication. In that model, the receiver models the sender to back-calculate the sender's state from received messages with transmission errors.

The abstract model, as presented in Chapter 2, is an extension of existing finite state machine methods. Mirage is also presented as an extension to timed Petri Net models in

Appendix F, although it may be applicable to any general instance of a compensator in a communication or control system.

Most of the relevant prior work on protocol models involves temporal extensions of finite state machines, such as timed Petri Nets and extended finite state models, and temporal logic. These all incorporate time as a boundary condition (event), record age as a static property of a variable, or denote only discrete time intervals, whereas Mirage considers the continuous effect of time on communication. These models also consider state as a point in state space and thus do not accommodate Mirage's extension to the use of probability density functions¹.

Mirage differs from FSM models in its use of time as a continuous parameter, in treating the remote space as indeterminate (as sets of possible states), in the dynamic use of partitionings of the possible remote state spaces, and in the use of such partitions to determine communicability and stability constraints.

3.1.1.1. Shannon's model of communication

Mirage can be considered to be an extension of Shannon's communication model [Sh63]. This is elaborated in Appendix A, but will also be briefly described here.

Shannon's model of communication is based on a point model of interaction. The state of the sender is a known point in state space, and the receiver interprets the arriving message as it refers to that point. This model describes error within the communication channel, of which a fundamental theorem is that the probability of error can be reduced to an arbitrarily small value, provided that the messages are encoded over a correspondingly arbitrarily large sequence.

Mirage can be considered an extension of this model, where Shannon's point value is a set in Mirage, and where point motion becomes set expansion, contraction, or translation due to various communication operations (Chapter 2). In Mirage, the conventional notion of connectivity and bandwidth is extended to account for latency, so that the connectivity of the communication graph is extended to denote topology, using link 'lengths'. Our model also extends the notions of effectiveness, correctness, and precision as described in Shannon's model, to temporal equivalents of timeliness, controllability, and communicability (see Appendix A).

¹Our use of PDFs is analogous to the analyses of state evolution of physical systems in thermodynamics.

Mirage is also a complement to Shannon's model, in several ways. Shannon's model considers a receiver's model of the sender, whereas Mirage considers the sender's model of the receiver. Shannon's error theorem trades error for latency, through sequence encoding, whereas Mirage trades latency for error (state imprecision) or restriction of operation (constraints).

3.1.1.2. Communicating finite state machines

Communicating finite state machines are a class of models for protocol analysis. These include communicating FSMs [Bo78], [Ok86], and FSMs extended with temporal constraints [Si82], [Ag83]. Neither case considers communicability. Extensions of these models are the basis for the Mirage protocol.

3.1.1.3. Petri Nets

Petri Nets [Pe62] are a variant of FSM models. The timed Petri Net model [Me76] can be extended to exhibit the characteristics of Mirage, just as the FSM model was; this extension is described in Appendix F. This demonstrates that the Mirage model is not restricted to a single paradigm.

3.1.1.4. Estelle / LOTOS

LOTOS [IS88a] represents a class of programming language models of protocols, which strictly includes Hoare's Communicating Sequential Processes (CSP) [Ho78], ISO's LOTOS, Milner's Calculus for Communicating Systems (CCS) [Mi80], and Real Time Attribute Grammars (RTAG) [An88], and which also includes hybrid programming language / state machine models such as Estelle [IS88b]. These models similarly lack expressions of communicability, although it may be possible to extend these models similarly. The most similar is RTAG, because the real-time scheduling constraints are a version of communicability, albeit on an individual message basis.

3.1.2. Partitioning the state space

Each of these traditional protocol analysis models is plagued by an explosion in the state space of the model [Bo78]. As the state of the analysis evolves, indeterminism of state requires modeling the powerset of possible states (i.e., all possible subsets). This

state space explosion is accommodated using imaging projections, factoring, and Powerdomains.

Part of the state space explosion is due to a set of messages in transit. FSMs cannot model a protocol with more than only a few messages in transit, and as such they cannot effectively model high information latency systems [Bo78]. The state space explosion can be managed by introducing state variables to factor some dimensions of the original FSM into variable-managing components.

By additionally partitioning the state variables, latency can be incorporated into the extended FSM [Si82]. This partitioning allows the separation of the effects of messages being emitted from a sender and collected by a receiver.

The state space can also be managed by partitioning, factoring, or imaging. Such partitions and projections may aid real implementations of the abstract Mirage model. The abstract Mirage model is intended for direct computation, but equivalent substructures of the model may be suitable. A more coarse-grained partitioning of the state space, using equivalence relations, accomplishes this. The subspaces become partitioned as a result of the imposed homomorphism of the relation, and particular location in the subspaces becomes less important than the traversal of these boundaries [Ch81]. This work was examined as ‘protocol conversion’, in both projections [La86] and general FSMs [Ok86].

3.1.2.1. Partitions

Partitioning a FSM is similar to the technique of transforming a nondeterministic finite automaton (NFA) into a deterministic one (DFA) [Sc82b], and to the partitioning principles of error detecting and correcting code analysis. Partitioning segments a FSM into independently verifiable components. Various methods determine an appropriate partition, including the use of artificial intelligence (AI) to differentiate between the explored and ignored components of state [Li87]. Mirage uses a similar partitioning in the communicability function, but ours is a dynamic process balancing partition and message sizes to ensure stable interaction among communicating entities.

3.1.2.2. Factoring

A FSM can be factored into its components. The reverse is more common, in which the product of component FSMs of a protocol generate the complete state space, factoring out statistically favored components. The remaining protocol is factored back down,

where possible. This results in a very efficient implementation, because a separate partition handles the most common FSM states, and a larger, more complex FSM is used only if necessary [Wo90].

3.1.2.3. Projections / imaging

Image protocols are formed by projecting functionality from the FSM. Given a specific function, an image of the FSM includes all states relevant to the provision of that function [La82]. These projections are similar to Mirage's modeling of remote state, which hides invisible state. They too can be extended to include latency and time [Sh82]. Projections also describe the partitioning of the state space in communicability, although in Mirage such partitioning is a dynamic process, whereas image protocols compute static partitions for analysis only.

Another version of imaging is found in entity models of protocols [Mo82]. These models consider the visibility of local state variables, in the same way as Mirage defines a perception.

3.1.2.4. Powerdomains

Mirage's analysis of state space subsets is similar to that of Powerdomains [Pl80]. Both Mirage and Powerdomains describe properties of the partitioning of state space into sets. Mirage bounds the size of the set of states as the system evolves; Powerdomains focus on the computability of the partition itself. A partition describing communicability therefore satisfies computability, because communicability requires a computable function to describe the evolution of the remote state space. Powerdomains consider set orderings as subset inclusion, whereas we consider a set ordering on size alone. These similarities are the result of the similarity between Dijkstra's guarded commands [Di76] (as analyzed by Powerdomains) and Mirage's guarded messages.

3.2. Protocol optimizations

The Mirage model can also be compared to model implications of specific protocol instances. Because Mirage addresses latency, especially in high speed wide area networks, comparisons focus on the model implications of light weight protocols

developed for similar domains [Do90], or on the model implications of flow control protocols for high speed networks.

Light weight protocols are developed by omission in functionality, hand-coded optimizations, or by limiting the protocol to a very specific domain. In contrast, Mirage is a general model for latency in communication, and it describes the optimizations used in light weight protocols in more general terms.

These light weight protocols include URP, VMTP, XTP, and versions of existing protocols (TCP) or variations that accommodate high bit latency (NetBlt). Many of these light weight protocols are instances of a general method we call ‘Cross Product Protocols’. New protocol designs have also focused on timers rather than packet exchanges, such as Delta-t, Virtual Clock, SNR (leaky bucket) and TP++. These protocols raise layering issues which Mirage avoids by its requirement of the predictability constraints of communicability.

3.2.1. Universal Receiver Protocol

The Universal Receiver Protocol (URP) [Fr89] is a data transport protocol that accommodates high bandwidth by a combination of omission and reorganizing the protocol components. URP assumes that the network will not reorder packets, and as such omits the reordering mechanism from the protocol specification. It also achieves high packet throughput by balancing the protocol load between receiver and transmitter.

One of the observations in URP is that conventional protocol code is unbalanced because most of the work is on the sender end of the protocol. Recent protocols mirror this imbalance, shifting the load onto the receiver (e.g., PROMPT) [Ba90a]. URP claims that a protocol is most efficient if the protocol work is evenly distributed between sender and receiver.

The Mirage model balances the actions of sender and receiver when the two interact evenly. The sender models the receiver, and anticipates its needs, whereas the receiver accepts and reacts to messages that correspond to its current state. In cases where the receiver can anticipate the reception of certain types of packets, Mirage would consider the data receiver to be modeling the state of the data sender. In Mirage the data flow direction doesn’t matter; state is modeled in either direction as desired, to the extent desired. The balance of the protocol depends only on the balance of the state modeling required.

Mirage's expansion of state space upon data transmission, and collapsing upon message receipt, has been examined in the design of buffer 'barriers' of URP [Fr89]. 'Barriers' is a modified flow control mechanism that attempts to equalize the uncertainty of communication among transmitter and receiver. Mirage's expansion and collapse of state modeling are the same as the expansion and collapse of buffer space allocation in barriers. Barriers apply to only buffer space state, and Mirage applies to the entire state of the remote node.

3.2.2. VMTP

The Versatile Message Transaction Protocol is a lightweight transfer protocol based on message transactions [Ch88b], [Ch89], [Ch86]. It provides a request-response capability, as in RPC (remote procedure call [Su88]), in addition to stream transmission, as in TCP. VMTP was "rethought from first principles," [Ch89], but these principles do not include domains of high latency, therefore much of VMTP is not applicable to Mirage's domain. VMTP reduces packet processing overhead and response latency, but does not mitigate the effects of transmission latency.

What is more important for future work in Mirage, VMTP combines concepts of RPC with NetBlt [Cl87] and flow protocols [Zh90]. Mirage may provide a model paradigm from which to unify these two datagram and data stream domains.

3.2.3. XTP

The Xpress Transfer Protocol (XTP) is another high speed transfer protocol [Ch88a], [Sa90]. XTP is designed for hardware implementation, and facilitates high speed communications, but was not designed for high latency.

XTP uses the exchange of state to manage rate control monitors, controlling the size and spacing of packets. The integral flow control mechanism manages the state exchange, and uses timers to force the exchange of state to reconnect after some types of failures. In the Mirage model, state is managed explicitly and the exchange of state is specified by imprecision, rather than on certain (error) events.

3.2.4. TCP

In Chapter 2, Mirage was analyzed with an extension of the conventional stream-based model of communication that the Transmission Control Protocol (TCP) provides [Po81a]. TCP supports only linear streams of data, whereas Mirage models branching streams as well. Protocols designed using Mirage reduce to TCP where the remote state is precise, i.e., where the stream does not branch.

TCP's sliding window flow control expects a linearity of the data stream at least as long as the window size requires, i.e., the round trip bit latency. Due to connection management overhead, TCP works efficiently where the data stream has a linearity that is at least an order of magnitude greater than the round trip time.

TCP has been extended for high latency domains by extending the maximum possible window size [Ja88a], and by modification of the windowed flow control feedback algorithms (also discussed later) [Ja88b]. These modifications assume that the round trip bit-latency can be filled with deterministically predicted data; Mirage assumes this is not the case, and that state imprecision will prevent effective utilization of the channel as a result.

Optimized TCP implementations have resulted in connections of 720 Megabits/second, demonstrating that TCP can operate in high speed networks [Ni91]. This verifies our original argument (Chapter 1), however, because these rates were achieved over very short (1-2 meters) distances (500 Megabits/sec), and in loopback mode (i.e., zero distance) (720 Megabits/sec). High speeds increase bit-latency, so because TCP works on a 45 Megabit WAN (the Internet), it should work equally well (modulo technological advances) at rates up to 10 Gigabits/sec on a MAN and up to a 100 Gigabits /sec on a LAN.

3.2.5. NetBlt

NetBlt is a variation of data transfer protocols optimized for bulk data transfer [CI87]. NetBlt assumes that the data stream has a linearity that is an order of magnitude longer than TCP does, and so is generally less applicable to Mirage's domain than TCP is. NetBlt optimizes bulk transfers by amortizing packet and acknowledgment overhead over large blocks of data, and adds a pacing (rate control) mechanism. This pacing

mechanism is contained in the Mirage model description, i.e., the sender emits packets when its model of the receiver determines that a receive buffer is available, according to known state of the receiver and elapsed time.

3.2.6. ‘Cross Product Protocols’

Another means to adapt existing protocols to high speed networks is a method of optimization we call ‘Cross Product Protocols’ (CPPs, for short). This includes methods of header prediction as in URP [Fr89] and TCP [Cl89], and methods of protocol layer merging and reduction as in XTP [Ch88a] and VMTP [Ch88b]. Header prediction and layer merging are both in the ‘protocol bypass’ [Wo90]. All these methods are included in the general method we call CPP.

The basis of a CPP is that the entire functionality of a protocol, either in implementation or layers of specification, is derived by a cross product of its components. Transfer protocols that implement a combination of the OSI transport and network layers are one example; protocols that combine features of RPC and streaming data transfer are another.

Indicated states are removed from the resulting protocol and treated as special cases, and the remainder of the protocol is factored down where possible. These states can be statistically favored (by direct measurement) [Kr85], selected to favor particular protocol operations, as in the ‘protocol bypass’, or determined by the current protocol state, as in the header prediction mechanisms of URP and an optimized TCP. The remaining states can be included in the resulting protocol, or removed to favor a ‘lightweight’ implementation (i.e., XTP, VMTP).

CPPs optimize protocol processing speed, but do not compensate for high bit latency. They are similar in focus to the statistical aspects of Mirage, in which pdfs determine expected values of the remote state. In addition, a CPP also collapses the *implementation* of the ISO protocol layer model, whereas the Mirage model considers the entire layer stack as the protocol. The result is that a CPP instance demonstrates how layering inhibits efficiency by restricting interlayer interaction, just as the Mirage model describes that layering prohibits latency compensation by hiding information required for the communicability function (i.e., regarding the state evolution function). Mirage can include layering, but only where it does not inhibit communicability (i.e., doesn’t hide

information required for prediction), in a manner similar to that of ‘information flow’ protocol analysis [II87], in which information is hierarchically maintained.

3.2.7. Delta-t

The Delta-t protocol optimizes low latency response and high throughput stream transfer [Wa89], [Wa81]. The shared state of a connection is assumed from a default, and updated only when a connection is in use, thus avoiding explicit connection management. Implicit connections support datagram and RPC messages without connection overhead, and without unnecessary agreement of stream-oriented buffer, window, and rate values.

A connection is assumed to have a predefined state, unless a current connection record exists that supersedes that information. The role of timers in state management is also shown, i.e., that timers are required in all protocols in lossy environments. Mirage uses time information to determine perception evolution, and so timers are implicit in the model. A protocol designed using Mirage implicitly includes bounds on the state evolution, whereas Delta-t enforces bounds by functions expressing viable states, as in the functions that describe the retransmission strategy. The implicit state of an unused connection expresses the initial conditions for communication, i.e., there must *be* shared state to *maintain* shared state; Delta-t permits these initial values to be nontrivial, whereas explicit connection protocols consider unopened connections to express an ‘undefined’ state. Further, Delta-t describes the initiation and evolution of the connection management information, which Mirage considers the ‘protocol’ itself.

3.2.8. Virtual Clock

The Virtual Clock (VC) protocol emulates statistical multiplexing using temporal windows to control packet flow [Zh89], [Zh90]. It uses time-based state information of the receiver’s ability to process packets to regulate the sender’s scheduling of packet emission. VC’s control of receiver buffer usage via sender constraint equations thus exhibits aspects of Mirage’s temporal evolution function.

There are aspects of the VC protocol to which Mirage will be applied in the future, notably the resynchronization of the clocks. As time progresses, the sender accumulates rights to send packets, assuming that it will use those rights and that the receiver will be processing them. If the sender does not use these rights, they expire. VC removes

transmission access rights by the periodic resynchronization of the clocks. This is similar to expansion of the state in the Mirage model, and a constraint that the space will not expand beyond a determined bound.

3.2.9. SNR (leaky bucket)

The SNR ‘leaky bucket’ protocol is a transfer protocol based on periodic exchange of state [Ne90], [Sa89], whereas comparable protocols (NetBlt, VMTP, XTP, Delta-t, etc.) exchange state primarily at the occurrence of an event. Because the complete state is communicated asynchronously to such events, SNR is self-correcting, i.e., it recovers gracefully from corruption and loss errors without additional mechanisms. Whereas SNR claims to be self-stabilizing (with proofs forthcoming, [Go]), Mirage sets stability as the criterion for state information emission, so protocols designed using Mirage are stable by construction.

SNR is a linear data stream oriented protocol that incorporates amortization methods as in NetBlt, and minimizes packet processing overhead as in VMTP. Amortization assumes an increase in the expected linear data stream length, and prevents branched stream utilization. The protocol is simplified by the use of a single timer mechanism to maintain state, resulting in reduced overhead. Further, both rate and window based flow control are consequent to maintaining shared state, i.e., by sharing buffer availability status.

The exchange of complete state allegedly prevents the sender and receiver from being ‘out of sync’ due to high transmission latency. State information packets are sent in anticipation of their request to ensure management of the perception imprecision. This provides greater state synchronization than explicit request/response mechanisms and accommodates known latency, given deterministic remote state evolution. If the remote state evolves nondeterministically or if the latency is variable, the mechanism in SNR does not ensure synchronization of state. The reasons for this are described further in Chapter 4, in the analysis of the Network Time Protocol.

SNR causes the receiver to send state updates to the sender in a pipeline to keep the sender’s perception of the receiver as small as desired, whereas protocols designed using Mirage cause the sender to emit messages in anticipation of the receiver’s request. SNR can be viewed as a restricted implementation of Mirage’s anticipation mechanism, where the receiver in turn models the sender’s model of itself, and endeavors to constrain that

model sufficiently by anticipating its temporal expansion. For example, the sender models the receiver, but that model also includes expansion in its perception of the receiver. The receiver is in a stable state, and knows that the sender is not aware of that stability, so it pipelines state information, recollapsing the perception at the sender each time a message arrives there.

SNR uses periodic exchange of state as a tradeoff between bandwidth (assumed in excess) and reduced processing overhead, and yet uses selective retransmission for packet loss. As little as a single bit can govern flow control [Ra88], but SNR exchanges complete state both for redundancy (thus error resilience) and because excess bandwidth is available, preventing computationally intensive packet processing. In addition, if bandwidth is excessive, forward error correction should dominate packet loss solutions, with simple bandwidth-wasteful retransmission as a backup (i.e., go-back-N). Selective retransmission reduces bandwidth at the cost of processing, contrary to the other design decisions of the protocol; this indicates that the protocol takes advantage of reverse path bandwidth excess only. Using the forward path in a lossy way is not considered, whereas in the Mirage model it is considered a viable complement to latency.

3.2.10. TP++

TP++ [Fe90a] is an extension of TP4, the OSI Transport Class 4 protocol which provides error detection and recovery [Ro90]. It provides a combination of latency constrained, transaction (RPC), and bulk data transfer (data streaming) communication. TP++ uses multiplexing to reduce state information among virtual connections. Multiplexing is claimed beneficial in reducing bandwidth requirements, but detrimental where particular state information must be retrieved.

The Mirage model considers multiplexing to reduce state information harmful. Such multiplexing inhibits the determination of the communicability constraint, and prevents latency reduction by anticipation with guarded messages. The Mirage model assumes that bandwidth is not a premium, and should not be reduced in ways that restrict latency compensation.

One statement associated with TP++ but not explicitly in the available literature is that of “running until you run out of state.”¹ This (unstated) assumption of protocol design is the explicit basis for the communicability and stability constraints in Mirage.

3.3. Communicability

One of the fundamental characteristics of Mirage is the communicability constraint, which indicates the conditions under which fixed latency can be accommodated by a protocol. Communicability denotes the requisite partitioning of the remote state volume, the need for guarded messages, and the need to express the temporal evolution of the remote state. Communicability also specifies the conditions under which stability can be maintained.

The basis for communicability lies in traditional communication theory, of cybernetics and control theory. Its implementation is similar to constraint mechanisms in monitors and methods for real time systems.

3.3.1. Cybernetics and control theory

Mirage is most fundamentally based on notions of controllability from cybernetics and control theory [As56], [Wi48]. Cybernetics is the basis of stability, as extended in Mirage, and sender anticipation. Mirage’s use of entropy as a measure of information dates back to Hartley [Ha28], although originally attributed to John von Neumann. This was described in Shannon’s communication theory, whose relevance to Mirage is elaborated in Appendix A [Sh63].

Stability has three variations — equilibrium, cycle stability, and phase space stability [As56]. Equilibrium occurs when a system reaches a single unchanging state. Cycle stability occurs when a system enters a set of states that is subsequently never exited. Phase space stability occurs when a system enters a region of phase space that is subsequently never exited. All of these types of stability are traditional, as referred to by ‘stability’ in Chapter 2; Mirage adds another type, entropic stability, as a further

¹This was mentioned by David Feldmeier in a presentation on TP++ at IFIP Protocols for High Speed Networks, 1990.

extension of these, in which stability occurs when the volume of a region of phase space is constant, rather than the particular region being fixed.

In addition, the TreeStack structure (Chapter 5 and Appendix G) in particular and the perception that it represents are encodings of persistent phase space trajectories (PSTs). PSTs are used in control and feedback theory to determine the stability of a system, usually defined as the confinement of trajectories to some bounded region of phase space.

In Mirage, perception encodes pending PST information. A perception is an afterimage of the traces of the possible futures of the remote state, held for the duration of the latency, until feedback resolves imprecision in the PST. This explains the need in Mirage to extend the notion of stability to include volume stability as well as the traditional state stability. In traditional control theory, PSTs converge to a finite region, whereas Mirage ensures stability by bounding the size of the afterimage, regardless of the path of the PST itself.

3.3.2. Time

There exist models of time in protocols that are more closely related to that of the Mirage model [Sc82b]. Incorporating time as a valid period for each state of a protocol machine is similar to denoting the interval over which the expansion of the subspace is well-defined [Ag83]. The extension of this method that presents hold times of protocol states as cumulative distribution functions is similar to a time extrapolation of this state space.

Time constraints have also been added to conventional protocol models, including timed Petri Nets [Me76], protocol projections [Sh82], and finite state machines [Sh85]. Time boundaries have also been used to denote limitations on state expansion explicitly [Fe90b], as in Mirage. These constraints are usually expressed as boundaries, but some describe state as a distribution function [Ag83], as does Mirage.

The incorporation of time into protocols has thus far been limited to two methods, where time is modeled either by boundaries or by finite time-steps. In the former, time is denoted by " $T_1 \leq T \leq T_2$ ", for some T_1, T_2 . Actions occur upon violation of these boundaries, as in time Petri Nets, temporal logic, or time-out timers [Sc82b]. In finite time-step models actions occur at some time instant, where " $T = T_1$ ". In Mirage, time is a

fully parametric value. The values of all other entities may change over time, and no barriers or finite points exist. Time is a continuous entity, over which other entities vary.

3.3.2.1. Real time systems

Real time systems also exhibit constraint mechanisms, usually in either an analysis or runtime scheduling capability. Here ‘real time’ means that operations terminate within a predeclared deadline, as opposed to the less formal definition of speed, i.e., fast enough to support ‘interactive’ requests. Real time system description languages include Hoare’s Communicating Sequential Processes (CSP) [Ho78] and Milner’s Calculus of Communicating Systems (CCS) [Mi80]. Mirage includes a scheduling constraint notion in communicability, i.e., in the scheduling of packets during the latency.

3.3.2.2. Aging variables

The modeling of time as an aging variable [Sc82b] is not as general as that in Mirage. Time markers age, but other entities do not vary with time. Aging reduces the precision of an entity. The aging of other variables may also affect the aging of a variable, so the function of time on the variables is arbitrary and variables are interdependent. In the Mirage model, time transforms the subspaces arbitrarily; the transformation is known at the time it is invoked, but the model does not restrict the transformation a-priori.

3.3.2.3. Timers

Other systems model time not by a variable but by timers that govern error recovery as in TP++ [Fe90c] and Delta-t [Wa89], or state maintenance as in SNR [Sa89] or Delta-t [Wa89]. Mirage uses time to model the entire state space, not just flow parameters, error control, or recovery state.

Time can also be a boundary, as in Real Time Communication (RTC) [Fe90b], including statistical boundaries, as in Mirage. In RTC there are temporal constraint formulae as in Mirage, including those for connection management, buffer availability, delay bounds, and statistical properties of ‘saturation’ (real time deadline overflow, i.e., saturation of the scheduling mechanism). Rate based flow control is a side effect of this use of timers and constraint equations in RTC, and future research in Mirage may also show other aspects of existing flow control protocols to be side effects of maintaining the communicability and stability constraints.

3.3.3. Constraints

The notion of restricting a machine to operate only within the valid subspace is an extension of distributed/replicated database techniques, most notably read/write quorum strategies [Gi79]. In addition, there have existed designs for external monitors that maintain constraints on a system, one of which is called the Overseer [Fa76a]. The use of these environments or supplemental programs to warn of dead-ends, maintain locality, and restrict other programs to within some valid subspace is similar to the methods used here. Mirage differs in that it appears that these notions are central to the operation of the protocol, not external, supplemental constraining devices.

Common knowledge methods of constraint are also relevant to Mirage, including conventional common knowledge [Ha84], database knowledge as in quorum consensus [Gi79], and in protocol analysis [Go88]. Each of these uses individual inference based on group constraints, as applied to incomplete information about the group. Mirage uses individual constraints, but includes similar inference, although common knowledge does not include Mirage's anticipation mechanisms.

Common knowledge as applied to protocols [Go88] denotes knowledge as a superset of state, e.g., knowledge includes facts true in all possible states, whereas state may include particular facts which may be asserted or omitted (although not inhibited). Mirage's 'perception' models locally all possible states of a remote entity, so in Mirage knowledge is explicitly denoted by the modeling of state. For example, if some states in a perception require a message emission, and no other states inhibit it, the sender would emit the message guarded to be received by the indicated states only.

3.4. Anticipation

Prediction in cybernetics utilizes constraints with sender/receiver modeling [As56]. This permits regulation, defined as isolating an external observer from observing internal system effects. In Mirage, this corresponds to anticipatory latency accommodation as it isolates the communicating entities from observing the actual latency between them, i.e., from being adversely affected by the latency. Regulation, also referred to as homeostasis, reflecting cybernetics as focused on biological systems, also is described in the presence of error (i.e., latency), and in the presence of constraints on the anticipatory control capability (i.e., communicability).

Anticipatory feedback is defined as the action of a ‘compensator’ (sender) when the ‘effector’ (receiver) has a time lag in action. As noted therein, “the conditions of stability and effectiveness of anticipatory feedbacks need a more thorough discussion than they have yet received,” [Wi48] which we believe is just as true today.

Anticipation mechanisms that are similar to Mirage exist in operating systems and congestion control, as well as areas of client/server extensions and some recent discussions in protocol research.

3.4.1. Operating systems

Most versions of anticipation use a depth-first search (DFS) of the possible state space. Some extend this to a version of breadth-first search (BFS) that examines some paths of possible state, and either terminates all but the desired path [Sm89] or rolls back the state of the paths that fail (Time Warp) [Je85]. Mirage exhibits BFS anticipation without rollback, but the source of the indeterminism (of the BFS tree) is latency induced imprecision, rather than algorithmically explicit, e.g., Monte Carlo execution methods [Sm89].

3.4.1.1. Concurrent execution

Jonathan Smith’s dissertation addressed the speedup of BFS searches by distributed BFS execution, with successful paths preempting the execution of concurrent attempts [Sm89]. Mirage is similar in the use of excess resources to save time. Mirage differs from such a BFS search in that entire levels of the possible state ‘tree’ are accounted by messages, so no preempting of failed paths is required. ‘Sibling elimination’ collapses the ensemble of processors to a single valid member, whereas Mirage requires a more intricate TreeStack pruning to remove models of multiple possible remote states.

Smith’s method is similar to Mirage if the set of remote processors is considered a single remote entity; in that case Mirage’s state imprecision corresponds to Smith’s indeterminism of process execution time or termination. Smith’s method uses BFS to permit multiple processors to participate in latency reduction in ensemble fashion, whereas Mirage models temporal imprecision in the state of the remote node.

3.4.1.2. Time Warp

The Time Warp system uses multiple distributed processes, each looking ahead, with rollback providing resynchronization (i.e., Time Warping) [Je85]. Time Warp is a distributed system, but in one of our first discussions (Chapter 1) we describe that there is little difference between a distributed system mechanism and a protocol, for which Mirage is designed. Even though rollback is a common method of error recovery in distributed systems [Ra78], Time Warp uses rollback to permit the distributed processes to proceed into possible future paths. Rollback occurs when a particular future of a participant is inconsistent with that of some other participant.

Messages in Time Warp are modeled similarly in Mirage, with sent messages being modeled as being both received and not received, and resolution of state occurring at a later time. In Time Warp the local process examines a single DFS path into the future, whereas Mirage examines all paths in BFS order, avoiding rollback. Resolution in Time Warp occurs via rollback, whereas in Mirage it occurs when the modeled state collapses using the subtree selection mechanism of the TreeStack.

Time Warp differs also in that it deals with message loss as the source of state imprecision, rather than time lag, as in Mirage.

3.4.2. Congestion control

Anticipation is beginning to appear in congestion control methods as well. We distinguish general methods of anticipation from congestion control anticipation because the latter uses a restricted state space. Congestion control, also called flow control, is performed by feedback, timers, anticipation, or a combination of these. The relative merits of each system can be evaluated as compared to Mirage.

3.4.2.1. Anticipatory congestion control

Predictive congestion control (PCC) “predict(s) (the) behavior of the system at a specific time instant in the future” [Ko90]. Mirage assumes that propagation delays dominate the system, whereas PCC claims that switching delays dominate. PCC attempts to reduce the switching latencies caused by buffering. In either case, the systems are based on the notion that the “state of a node may change considerably between sending

messages (with control) and receiving them” [Ko90]. In Mirage, all messages control the state space, but the assumption that state varies during communication latency is the same.

Reactive congestion control fails due to the fixed latency, so proactive measures are indicated. PCC is similar in principle to Virtual Clock, albeit derived differently. PCC, like Delta-t and SNR, uses periodic exchange of data and rate control state to govern flow control. PCC is directed at individual hop flow control, but extends to end-to-end methods as well.

Mirage is also proactive, although as it regards the entire controlling state of a node, rather than just the flow of data. Mirage is thus a more general form of PCC, based on more general principles of state sharing and stability.

3.4.2.2. Timer-based congestion control

The Stop-and-Go flow control protocol reduces latency jitter in packet transmission [Go90]. This protocol is designed to be used where the state of the system (available buffers) is deterministic with respect to time, but where latency jitter induces the same imprecision in remote state perception as nondeterministic state evolution would have. This isomorphism is addressed in Chapter 4. Jitter is reduced at the expense of an increase in the minimum latency, which is advantageous only where the remote state imprecision is a function of jitter alone, i.e., where the remote state is otherwise deterministic. Such is the case in some time protocols, such as the Network Time Protocol [Mi90b], also addressed further in Chapter 4.

3.4.2.3. Feedback congestion control

Other mechanisms more indirectly infer feedback information for flow control [Ja89]. Jain’s mechanism uses round trip time variability to determine network overload, and to adjust packet flow accordingly, but this is a reactive method of control, in which adjustments occur after a round trip latency. Reactive flow control works properly on the time scale of tens of round trip times, whereas proactive control methods, such as in Mirage, are required for more spontaneous activity [Ja90].

Jain also argues for multidimensional flow control, noting that window vs. rate control, open vs. closed feedback, router vs. source controls, and reservation vs. ‘walk-in’ bandwidth allocation mechanisms each affect the entire scheme of flow. We consider that these are each an endpoint in a continuum, and that the entire set of controls denotes a

space of possible flow protocols, because each requires additional state modeling in Mirage. Conventional mechanisms investigate endpoints of the flow protocol space, whereas further investigation using Mirage may demonstrate protocol versions that exhibit any location in that space.

3.4.2.4. Combination control

Other attempts have been made to reexamine round trip time estimation, as it pertains to flow control and state variability, including those by Jacobson [Ja88b]. In terms of Mirage, state is the send window size and the rate parameter of the flow control, as managed by predictions of round trip time and altered by feedback information. Initial state is assumed to have a unit window and a minimal (one-time emission) rate, which is a minimal initial state resulting in sustained communication, as compared to the non-minimal initial state of the Delta-t protocol. Methods to adjust this state are expressed by the so-called Slow Start and Exponential Backoff algorithms, and the inverse exponential ‘round trip time’ sequence averaging. In the past, these algorithms have been considered extraneous to the declaration of the TCP protocol [Po81a], but Mirage indicates that the TCP protocol cannot be modeled without them, because these equations govern the evolution of state and thus communicability and stability.

3.4.3. Other forms of anticipation

Other versions of anticipation include extensions to client/server models, recent conference presentations in protocols, and most notably computer architecture. The similarity between methods of protocol anticipation in Mirage and computer architecture is described in detail in Chapters 5 and 6, in which we describe a novel architecture for processor-memory interaction using anticipation mechanisms in active memory architectures.

3.4.3.1. Client / server extensions

Existing client/server protocols exhibit traditional request/response protocols, such as Remote Procedure Call (RPC) [Su88] and the Network File System (NFS) [Su89]. Extensions to RPC include Remote Evaluation (REV), which sends the code to the data, rather than the conventional converse of RPC [St90]. A further extension of REV, called Late Binding RPC, provides for some kinds of anticipation [Pa91]. The difficulty with

RPC mechanisms is that the evolution of the perception of the remote space is governed by the function within the RPC. The RPC functions thus determine communicability, and their variation makes stability more difficult to ensure.

3.4.3.2. Recent protocol discussions

A recent conference in high speed protocols resulted in several current considerations which are similar to components of Mirage [Pa90b]. These include ‘parallel RPC’, network anticipation, and ‘asynchronous RPC’.

‘Parallel RPC’ (of Craig Partridge) requests data in excess of expected utilization, in the hope that some will be of use. Excess bandwidth is wasted to keep the CPU fed, in the prediction that CPU starvation will result from increased bit latency. This method is similar to the parallelization in Jonathan Smith’s method of distributed execution [Sm89], which is ensemble based, whereas Mirage is based on possible states of a single remote process.

‘Network anticipation’ (of Jonathan Smith) is a receiver version of sender anticipation in Mirage, and is similarly used to keep a starved CPU fed. This method is the network analog of conventional cache anticipation mechanisms, whereas Mirage suggests new versions of active memory anticipation in contrast. Sender anticipation in active memories is addressed in Chapters 5 and 6.

‘Asynchronous RPC’ is a new paradigm for interaction that accommodates latency (Thomas Joseph). He notes that asynchronous RPC may require larger messages, but that wasting bandwidth to permit asynchronous return frees the local processor during response latency. A-RPC is not anticipatory, as Mirage is, but it does provide a mechanism in which Mirage’s analysis may also prove useful.

David Cheriton also presented a talk recently [Ch91] about ‘dissemination oriented communication’ (DOC). It describes an eager communication system, in which round trip latency is avoided for conditional behavior. This system is similar to a SIMD¹ conditionally executed instruction, i.e., where opcodes are executed or ignored depending upon local processor state; messages are sent and accepted only if the receiver is in the proscribed state. Such messages are the same as Mirage’s guarded messages, although the guard performs different functions in each. DOC is geared towards conditional execution

¹Single Instruction Multiple Data, denoting a multiprocessor system in which opcodes are broadcast to the processors, each of which operates on its own data stream.

in an ensemble of processes, as in the SIMD system, whereas Mirage models imprecision in knowledge of a particular state (i.e., temporal imprecision).

3.5. Physics analogs

Mirage was originally developed from analogies from physics, as described in Appendix B. The analogies are based on the multiple-worlds model of quantum interaction, and the Feynman path integral method for particle interaction analysis. The multiple-worlds model is also similar to those in Truth Maintenance Systems, which describe data structures for maintaining simultaneous logical states, similar to the TreeStack structure developed for the μ -Net architecture presented in Chapter 5, and discussed in Appendix G.

3.5.1. Truth Maintenance Systems

The term ‘Truth Maintenance System’ (TMS) was coined by Jon Doyle, and refers to a logic system that maintains possible logical states [Do77], [Do79]. This is similar to the sender’s perception of the receiver as modeled as a set of possible states in Mirage. In a TMS, this set of states is modeled by a data structure, which is maintained by a combination of instantiating extensions and collapsing revisions; in Mirage, the set is expanded upon message emission, and collapsed upon message reception. The perception in Mirage has been extended to model recursion in the remote state (in μ -Net, in Chapter 5), but it is not clear whether a corresponding recursive TMS exists, although recursion is used in TMS methods.

3.5.2. Physics in protocols

The Computational Field Model (CFM) is recent research in the direct application of physics analogies to distributed systems issues [To90b]. CFM is an “object oriented open distributed environment” that addresses the issues of computing processes (i.e., programs) affecting the environment of the computation, permitting redistribution of the processes. CFM also notes that message passing and object oriented hidden messages are

not capable of modeling communication latency; instead, a model of “assimilation / dissimilation” is developed.

CFM defines distributed computing as parallel computing with interprocess communication delay, similarly to our definitions of communication in the Introduction (Chapter 1). *Distance* is defined as geographic distance, under the presumption that this is proportional to communication latency. *Mass* is defined as the size of the object, and *inertia* as the migration overhead. These definitions are not otherwise justified, i.e., inertia is necessarily linearly related to mass, as in their physical analogs.

Gravitational force is defined as the bandwidth of communication (size and frequency of communication, therein). A counteracting *repulsive force* is defined as “the product of the size of the objects over the distance to the n-th power.” No rationale for the definition of these forces is given. Gravity should be a benefit function that causes objects to migrate together. Repulsion is a cost function causing objects to migrate apart.

A message is considered a “special object,” whose migration is handled through the addition of primitives of dissimilation, migration, and assimilation primitives. It is not clear why a more unified approach is not used, where messages are simply objects that are defined as having an infinite repulsion to the sender and an infinite gravity to the receiver. *Assimilation* is message creation and *dissimilation* is message absorption, which are required in either case to describe object ‘fission’ and ‘fusion’ (terms we prefer, because they imply messages and nodes are the same ‘matter-energy’). In CFM, delivery service classes and timeouts can be specified in terms of message inertia, mass, and lifetime to decay, rather than introduced as separate characteristics of migration.

The CFM bears a resemblance to our original model basis, in which communicating entities are described as particles, and communication is described as particle exchange. Our original model, however, does not differentiate messages from endpoints, and presents a more unified approach that we feel more accurately adapts physics analogs to communications principles. This may be the effect of our choice of utilizing the particle exchange description of physical forces, rather than fields, described in detail in Appendix B. Particle exchange seems more accurate because messages represent particles. The field model of forces is likely to be more appropriate to repulsive cost description, because multiple objects should be prevented from sharing a single processor. This latter field is a binary function, because there should be no difference in the repulsion between objects on two adjacent vs. two separated processors; other costs of

interaction (communication) should govern their attraction, or permit them to move about independently.